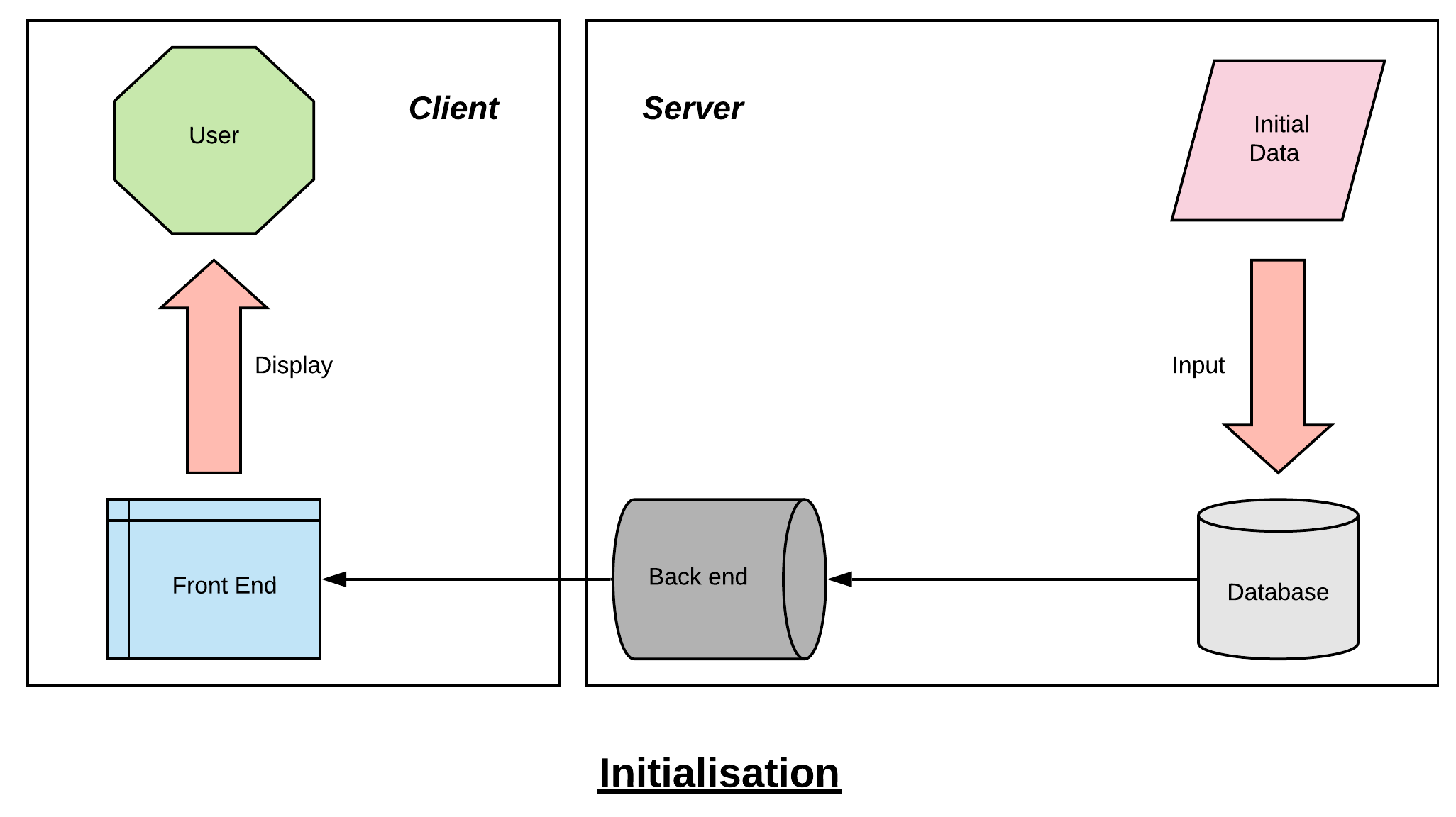
Requirement specifications

***General introduction***

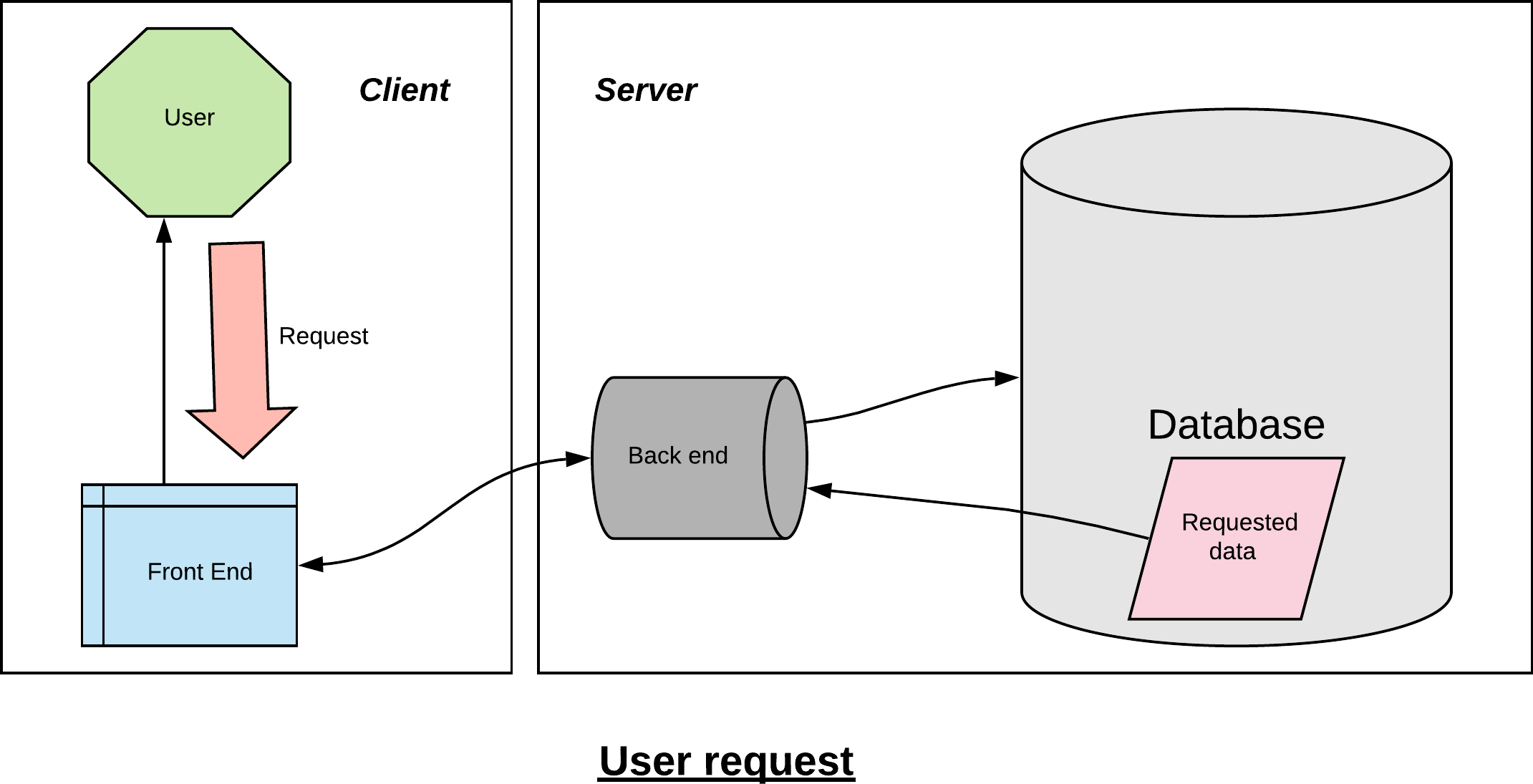
On the European scale, 41 electricity network companies take care of the smooth functioning and the reliability of the European electrical network. RTE is the French electricity network company. Currently, we use the “N-1” method, i.e. we verify the proper network functionment whenever we switch offline one component of the network. However this is an incomplete approach to the problem as it does not enable an adequation with the context. This could generate under-efficiency in a favourable climate, but also it could not ensure a high level of reliability when the climate is unfavourable, as in tempests or floodings. This means the “N-1” method does not distinguish weather conditions, even if bad ones generate troubles on the network.

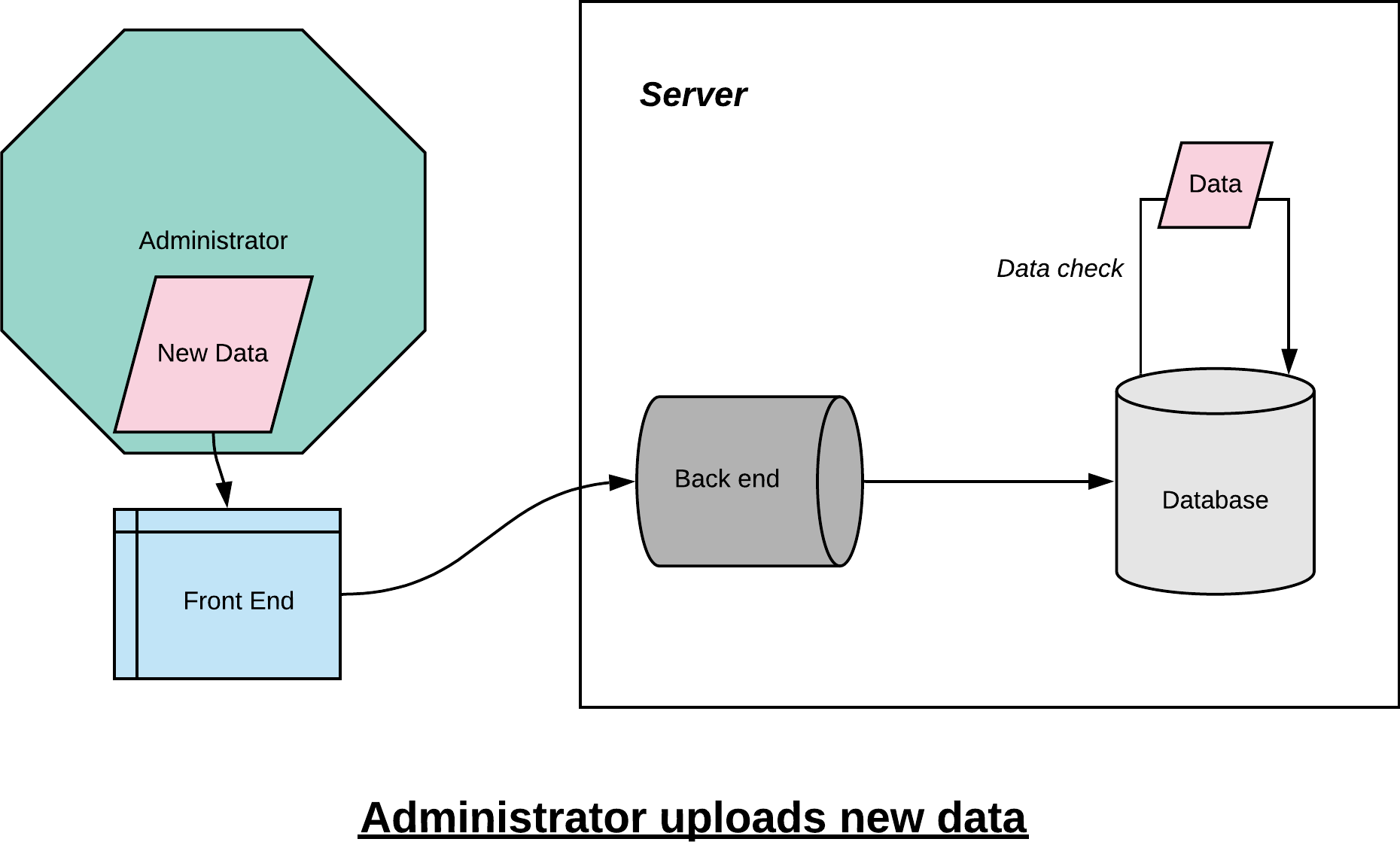
This data science project consists in the implementation of a structure enabling the collection of disturbances on the electrical network on the European scale, by the creation of a database to keep a record of all disturbances in a normalized form and the creation of an HMI (Human Machine Interface), which enables anyone to freely browse through the data currently stored in the database, but also to input new data into the database for experts. Then statistics will be computed by this database, in order to get a better understanding of disturbances and at last avoid them. In the rest of this document we are going to give our interpretation of the problem at hand, and explain what we are going to do to implement an adequate solution to this problem.

***Presentation of the project structure***

First of all this program is created and structured as depicted in the “Initialisation” below. The server should include a database, in which all available data from previous experiments such as Nordel, but also a back end server, through which all interactions with the database should be. This back end server will then send all informations to front end servers on all users’ computers, for them to be displayed.

Then once this structure is created, the system will act differently towards its users, based on their profile : client or administrator. Clients are only allowed to send requests to the database through the back end server, that will select needed datas and send them to the client.

Administrators can also send requests to the server in order to visualize some data. However they are also allowed to upload new data into the server, through their personal front end server on their computer. This data will be checked and put into the adequate format before being included in the database.



***Functions listing***

* db\_init(name)
* Initialization of the database with a given name. The specific structure of the database and data stored inside are already well defined according to the preconisation.
* db\_insert\_disturbance(db, disturb)
* Takes as parameter the name of the database and the disturbance character strings in a form to define (tuple in python) and adds this disturbance to the database. Returns an error if the disturbance format is not the correct one. If the disturbance identification already exists, returns a different error and asks if there is a need to replace the old record.
* db\_insert\_fault(db, disturb, fault)
* Takes as a parameter the name of the database, the disturbance identification number and a fault character string in a form to be defined (tuple in python) and adds this fault to the database. All parameters required are stored in a tuple named “fault”.
* db\_insert\_outage(db, outage)
* Similar function as above but to add outages. All parameters required are stored in a tuple named “outage”.
* db\_insert\_interruption(db, disturb, interrupt)
* Similar function as above but for inserting interruptions. All parameters required are stored in a tuple named “interrupt”.
* db\_select\_disturbance(db, disturb)
* Takes as a parameter the name of the database and an identification number of a disturbance in a form to be defined (string in python) and searches for this disturbance in the database. Returns all the associated records (disturbance, faults, outages and interruptions) if the search succeeded or an error if the disturbance identification number is not found.
* db\_update(db, disturb, faults, outages, interrupts)
* Takes as parameters the name of the database and strings in a form to be defined (tuple in python) and updates this disturbance in the database.
* db\_delete(db, ident)
* Takes as parameters the name of the database and the identification number of a disturbance in a form to be defined (string in python) and looks up this disturbance in the database, then deletes all related records if the search succeeded or returns an error if the disturbance identification number is not found.
* db\_output(db)
* Takes as parameter the name of a database and returns all saved information in a format to be defined
* HMI\_access(db\_users, id, pw)
* Takes as parameters two strings and returns:
* 0 if the id is not in db\_users
* 1 if the id is in db\_users but isn’t associated with the password pw
* 2 if the id is in db\_users and associated with the password pw

***Database***

The database will be used to store information. Disturbances, faults, outages, interruptions and related information will be stored in different charts in the database. To be adapted to the grammar of SQL-databases, most information will be stored as a text, including date and time.

In the initial stage of project development, SQLite will be used for local tests. In the first step, the database will be created and formatted according to the requirements. In the second step, new records will be generated based on all incoming data and inserted into the corresponding chart of the database. The third step is to implement the information retrieval function, that searches and returns the corresponding information in the database according to the given identification numbers, and send them back to the front end.

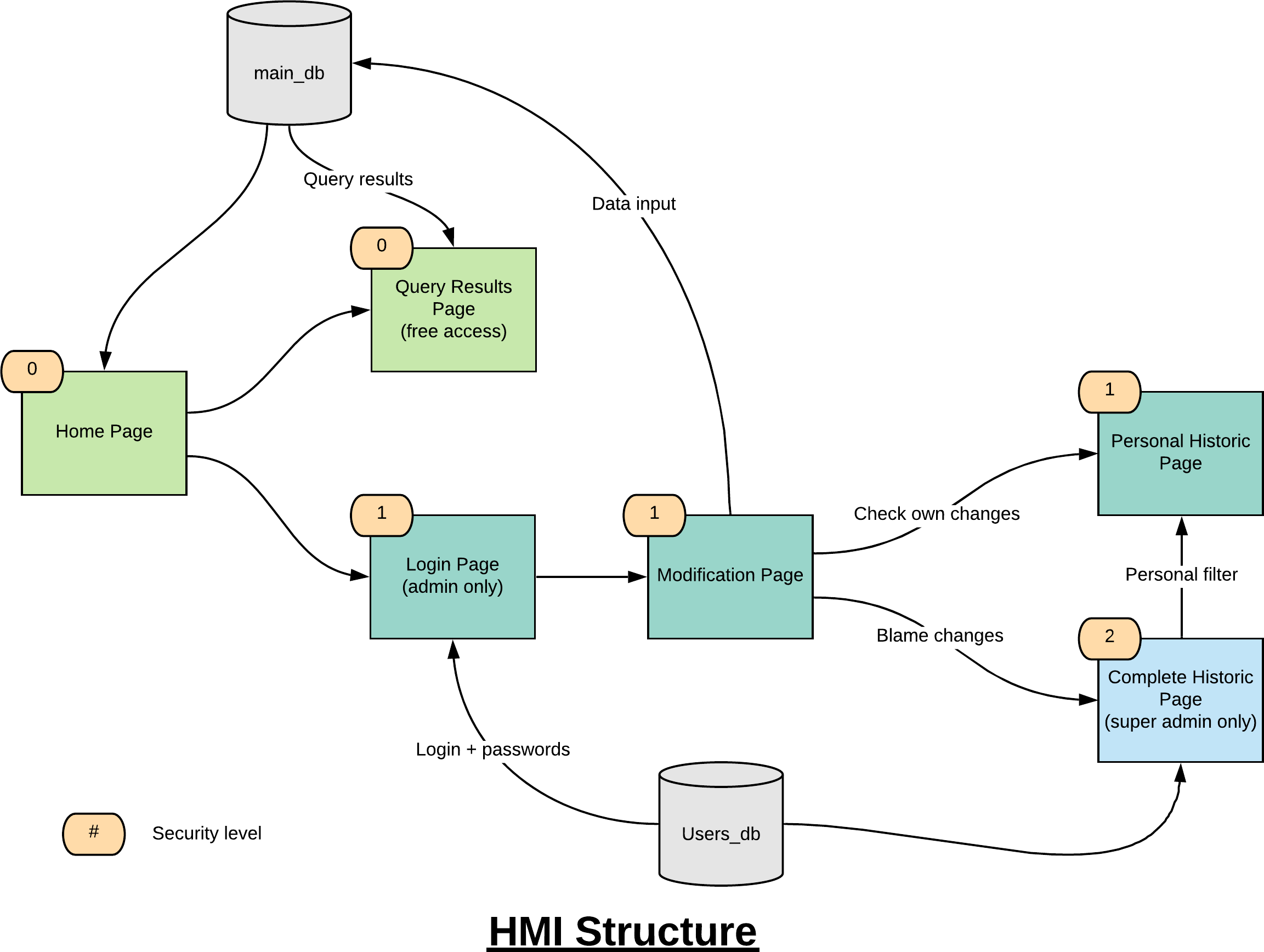
Then MySQL will be used, and the official database for the server will be implemented in this format. Compared to SQLite, some more advanced features will be added. For example, allow multiple users to connect to the database at the same time, and manage the access rights of the database by username.

During the use of the MySQL database, changes to the database will be recorded. According to the access rights of different users, they can query and browse through historical records within the allowable range : normal administrators may see the historic of their changes, whereas super administrators are allowed to see the full historic of all changes.

In the next section, we will explain how to connect the server-side database with the user-side through HMI.

***The HMI organization***

The HMI will be organized in the form of several layers of pages presented in figure 1. For security issues we will create two separate databases : main\_db that contains all information of the network and users\_db with all users’ information (mainly IDs and passwords). Finally, the horizontal arrows in the figure show the information flows between the HMI and databases.

The functionalities and features of each page are detailed in Table 1.

*Table 1: HMI features*

|  |  |  |
| --- | --- | --- |
| ***Page*** | ***Features*** | ***Details*** |
| Home | Searching | Users enter a given string and the HMI shows the results page with all database lines containing the input |
| Filtering | TBD |
| Accessing the login page | A login button that links to the Login page |
| Results | Saving | Users could save the results as CSV (or pdf) |
| Returning | Enabling users to return to the Home page |
| Login | Logging in | A classic feature with two entry zones : one for the ID and the other for the password |
| Modification | Inserting a new object | Users could create a new object by choosing its type (disturb, fault…) and entering the required parameters |
| Searching, filtering and saving the results | Same as the Home page |
| Deleting an object | Users could select lines from the results and delete them |
| Changing the value of an object | Users could change the value of a parameter directly from the results |
| Saving changes | A button to confirm and save modifications |
| Logging out | A logout button that links to the Home page |

**Automated tests**

Each implemented fonction needs to pass automated tests to ensure its functionment. Automated tests will also be set up for the addition, the deletion or the modification of an element.

These tests need to be written before the corresponding code, once the name of the function is set. This step-by-step approach will ensure the smooth proceeding of the code. We will execute these tests thanks to the pytest framework.

To ensure the well functioning of the HMI, we will use the examples given in the sixth chapter of “Guidelines for the classification of grid disturbances above 100kV”.

Let’s give examples of test that could be realised :

* def test\_add\_disturbance():  
   disturbance = “2016-1;2016-01-10;10:01”  
   db\_insert\_disturbance(db,disturbance)  
   query = “““ SELECT \* FROM disturbance WHERE id=”2016-1” ”””  
   disturbance\_0 = execute\_read\_query(connection,query)  
   assert len(disturbance\_0)==1  
   dist=disturbance\_0[0]  
   assert dist[0]==”2016-1”  
   assert dist[1]==”2016-01-10”  
   assert dist[2]==”10:01”
* def test\_HMI\_access():  
   users\_query = “”” SELECT id,pw FROM users “”””  
   users = execute\_read\_query(connection,users\_query)  
   if len(users)>0:  
   user\_id = users[0][0]  
   user\_pw = user[0][1]  
   assert HMI\_access(db\_users,user\_id,user\_pw)==2  
   assert HMI\_access(db\_users,user\_id,user\_pw +”an error”)==1  
   assert HMI\_access(db\_users,”Not a Name”,”Password”)==0