" Document for database manager system "

First we use the factory design pattern and the singleton
design pattern. I will show it later .

now let's start with main :

here i make access form main to 3 files: Connection.js
file , QueryController.js file and FileController.js file To
get to the functions in them .

```js
const Connection = require('./Connection.js')
const QueryController = require('./QueryController.js')
const FileController = require('./FileController.js')
```

then import the Connection class from file Connection.js :

```js
let connection_ = new Connection.Connection()
```

After that, I give the system the Shared information to
make connections at any Postgres database .

ConnectionDetails_ is an object. I declare it in the
connection.js file and I import it here to set its values
.

```
let connectionDetails_ = Connection.connectionDetails;
connectionDetails_.hostName = prompt("enter the name of host : ");
connectionDetails_.user = prompt("enter the name of user : ");
connectionDetails_.passWord = prompt("enter the password : ");
connectionDetails_.port = prompt("enter the port : ");
```

here i call GetNewConnection function from Connection class to
connect with database in postgre sql .
the parameter we need it to make a new connection is connection
details and name of database .

the connection on local database is done successfully
here

```
const localDB=connection_.GetNewConnection(connectionDetails_,"localDB");
localDB.connect()
```

I need to make a connection on the local database for
connection profile and to authorize the user at sign in
process .

here we have the main function which we will calling it
to finally :

```javascript
async function Main(connectionDetails_){
  const log4js = require('log4js');

  // Create the logger
  const logger = log4js.getLogger();
  try{
      console.log("Sign in :")
      usrId =prompt("enter your id  : ");
      passWord =prompt("enter your password  : ");
      const result =await autherizUser(usrId,passWord);


      if (result.length == 1){
        logger.level = 'info';
        logger.info('Successfully!');
        console.log("welcome in your manegar database : ")
        database = prompt("enter the name of DB : ");

        // Instantiate User
        connection_.ConnectionProfile(connectionDetails_,database,localDB)

        const client=connection_.GetNewConnection(connectionDetails_,database);
        client.connect();



         await Implementation(client)
      }
      else if(result.length ==0){
        logger.level = 'error';
        logger.error('The id is not available...');
        console.log("sign_up plz")
        return 0;
      }

  }
}
```

We import log4js for loggers .
First the user will enter his/her id and password to make
authorization to the user .

after the user enter her/his id and password and send it
to
```javascript
const result =await autherizUser(usrId,passWord);
```


this function will make quere at localDB at users table
particularly at user_id column if this table has user id
and password that's matching with id and password which
entered by user then its return result .

```
function autherizUser(userId,password){
    return new Promise( ( resolve, reject ) => {
        localDB.query("SELECT user_id FROM users WHERE user_id = '"+ userId +"' and password = '" +password+"'", (err
            if (err){
                return ( err );
            }
            resolve(result.rows);

        });
    })
}
```

if the length of result from AuthorizedUser function ==1
that means that we have a user with an id and password
like the entered id and password .

```
if (result.length == 1){
        logger.level = 'info';
        logger.info('Successfully!');
        console.log("welcome in your manegar database : ")
        database = prompt("enter the name of DB : ");

        // Instantiate User
        connection_.ConnectionProfile(connectionDetails_,databa
se,localDB)

        const
client=connection_.GetNewConnection(connectionDetails_,database);
        client.connect();


         await Implementation(client)
    }
    else if(result.length ==0){
        logger.level = 'error';
        logger.error('The id is not available...');
        console.log("sign_up plz")
        return 0;
    }
```

if the authorization is done successfully .
Now the user will be able to use the services provided by
the application .

enter the name of the database which the user wants to
make a connection on.
after enter the database name we will call the
ConnectionProfile
function to store the base on connections table if it is not
founded or to make the connection directly .

```
database = prompt("enter the name of DB : ");

        connection_.ConnectionProfile(connectionDetails_,databa
se,localDB)
```

```
const
client=connection_.GetNewConnection(connectionDetails_,database);
client.connect();
```

the const client is connected to the database the user
choice it .

finally we will call the `Implementation(client)` function
and send to client to making query on it .

in the `Implementation` function we print a list of services we
providing it for user .

```
        console.log("\n      List of Jobs :) ")
        console.log(" 1. make a query :) ")
        console.log(" 2. import file to database :) ")
        console.log(" 3. export data from database :) ")
        console.log("........................................
")
        let choice = prompt("enter the number of job you want
from list : ");
```

And ask the user to enter any service he wants?

if the user choose 1 (make a query):
another list will display on console ::
And ask the user to enter any service he wants again ?

```
        case '1' :
         {
          console.log(" List of query :) ")
          console.log(" 1. write your query :) ")
          console.log(" 2. Get Table Data by add table name :) ")
          console.log(" 3. Get tables in database :)***********")
          console.log(" 4. Get culumns for spesfic table :)***********")
          console.log(" 5. Drop Table by table name :) ")
          let choice_ = prompt("enter the number of job you want from list  : ");
```

 Here we ask the user to enter the type of query he want to to
send it to the GetInstance function to have instance from this
query type and store it on the queryInstance variable ..

```
          let choice_ = prompt("enter the number of job you want from list  : ");
          let queryController= new QueryController();
          let queryType = prompt("enter the type of query you want ('Insert','Read','Write','Delete')  : ");
          let queryinstance =  queryController.GetInstance(queryType);
```

if the user choose 1 :
     he/ she will write any query he/she want

but to make the experience easier without the front end I print
all tables on the database by calling the GetDBTables function
and send the client which connects to our database .

then call GetQuery(client) function .

```
const result = await queryinstance.GetDBTabels(client);
console.log(result)
queryinstance.GetQuery(client);
```

GetQuery(client) function :
it asks the user to enter its sql query then send it to Api to
get the result .

```
GetQuery(client){

    let query=enter("enter your query plz : ")
    client.query(query, (error, result) => {
        if (error) {
            ogger.level = 'error';
            logger.error('syntext query error');
            console.log(error.stack)
        } else {
            logger.level = 'info';
            logger.info('show the rows');
            console.log(result.rows)
        }
    })
}
```

if the user choose 2 :
first all tables in database will display for users to select the
table he want to display it data .

```
const result = await queryinstance.GetDBTabels(client);
console.log(result)
queryinstance.GetTableData(client);
}
```

This is an addition by me "dina" .

first i ask the user enter the table name and then send it to api
to display row of data in this table ..

```
    //Dina
    GetTableData(client)
    {


        var tableName = enter("enter table name : ")
        client.query("Select * from "+tableName, (error, result) => {
            if (error) {
                logger.level = 'error';
                logger.error('the data is not found');
                console.log(error.stack)
            } else {
                console.log(result.rows)


            }
        })
    }
```

if the user choose 3 :

the system will call GetTabels(client)

```
    try{
        const result = await queryinstance.GetDBTabels(client);
        console.log(result)

    }
```

GetTabels(client)

```
    GetDBTabels(client){
        return new Promise( ( resolve, reject ) => {
            client.query("select table_name from information_schema.tables where table_schema='public'", (err, result)
                if (err){
                    logger.level = 'error';
                    logger.error('the table is not found');
                    return reject( err );
                }
                {

                    resolve( result.rows);

                }

        });
    })
```

if the user choose 4 :

a list of all tables will display for user then by call GetTables
function and then we ask user to enter table name to send it for
GetColumns function to return the result for user and display it
on console .

```
const result = await queryinstance.GetDBTabels(client);
console.log(result)
const tableNname = prompt("enter name of table you want to display its columns  : ");
const columnsResult = await queryinstance.GetDBColumns(tableNname,client);
console.log(columnsResult)
```

if the user enter 5 :

the user will enter the table name he/she want to drop it and
send it to the DeleteTable function .

```
const result = await queryinstance.GetDBTabels(client);
console.log(result)
const tableNname = prompt("enter name of table you want to display its columns  : ");
await queryinstance.DeleteTable(tableNname,client);
```

this is the DeleteTable function .

```
DeleteTable(tableName,client) {

    client.query("drop table "+tableName, (error, result) => {
        if (error) {
            console.log(error.stack)
        } else {
            console.log("table is deleted : ")


        }
    })
}
```

if the user chooses 2 (import file to database)from the first list ..

the system ask the user to enter the file type to send it to ImportFile at FileController Class which I have create instance from it .

```
case '2':
    {

    let fileType = prompt("enter the type of file you want import to Database  : ");

    const fileController_ = new FileController();

    fileController_.ImportFile(fileType,client);

      break;
    }
```

if the user enter Csv in ImportFile at FileController Class :

we will call GetInstance and send file Csv string as parameter to send it to createInstance at FileFactory .

then i read the file by call ReadFile function witch ask the bath of the file to read it .

After reading it and store the result at data variable i take the header at index 0 and send it to SchemaIsFounded function ..
to check the schema of all tables in the database with the schema for csv file and the return table name if it is found or return null if it is not found ..

if the table name is not equal null i will send the data without header to the ImportData function , Which will roll over all rows in   a   data   ,   and   send   it   to   the InertFilesData(values,table,client) function at Query.js file to implement post Api thats will insert the data to table.

else if the table is not founded i will call creatTable function in Query.js fill witch ask the user to enter table name and take csv file schema and create new table then call ImportData function to fetch the data to table .

```
if(fileType==fileType__.CSV){

    const csvFile=this.GetInstance(fileType)
    let data =csvFile.ReadFile("C:\\Users\\hp\\Desktop\\all\\data.csv")
    csvFile.SchemaIsfounded(client,data[0],fileType).then(tableName => {
        console.log(tableName)
        if(tableName!=null){
            data.shift()
            data.pop()
            csvFile.ImportData(data,tableName,client);
        }else{
            let schema = csvFile.CreateSchema(data[0]);
            console.log(schema)
            tableName =insert.CreateTable(schema,client);
            data.shift()
            data.pop()
            csvFile.ImportData(data,tableName,client);
        }
```

SchemaIsFounded function :

```
async SchemaIsfounded(client,data,fileType){
    let result;
    let get=new query.Read();
    const dataBaseTables =await get.GetDBTabels(client);

    for(var i=0;i<dataBaseTables.length;i++){

        const columnObject= await get.GetDBColumns(dataBaseTables[i].table_name,client);
        var columnArray = [];
        for(var j=0;j<columnObject.length;j++){
            columnArray.push(columnObject[j].column_name)
        }

        if(JSON.stringify(columnArray) === JSON.stringify(data))  {
            result=dataBaseTables[i];

            logger.level = 'info';
            logger.info('schema has found');
                return(result.table_name) ;



        }

    }
}
```

ImportData function :

```
ImportData(data,tableName,client){

    for (var i = 0; i < data.length; i++) {
        var values=this.GetValues(data,i)
        var table = tableName ;
        const insert = new query.Insert();
        insert.InertFilesData(values,table,client)
    }


}
```

the same thing for the Json file ...

```
        }else if(fileType = fileType__.JSON){

            const jsonFile=this.GetInstance("Json")
            let data =jsonFile.ReadFile("User.json")

            let jsonHeader=[]
            for (let key in data[0]) {
                jsonHeader.push(key);
            }

            let row_=[]
            let rows_=[]
            for(var i=0;i<data.length;i++){
            for(var j=0;j<jsonHeader.length;j++){
                row_.push(data[i][jsonHeader[j]]);
            }

            rows_.push(row_)
            row_=[];
            }

            jsonFile.SchemaIsfounded(client,jsonHeader,fileType).then(tableName_ => {
                console.log(tableName_)
                if(tableName_!=null){

                    jsonFile.ImportData(rows_,tableName_,client);

                }else{

                    let schema = jsonFile.CreateSchema(jsonHeader);
                    let tableName_ =insert.CreateTable(schema,client);
                    jsonFile.ImportData(rows_,tableName_,client);


                }
```

then if the user choose 3 (Export date from table to file) :

We will write the data from a specific table in the data table
with its header to the csv file .

```
18   // open   PostgreSQL connection
19   pool.connect((err, client, done) => {
20    logger.level = 'Error';
21    logger.Error('connection failed');
22    if (err) throw err;
23
24     client.query("SELECT * FROM customers", (err, result) => {
25       done();
26       if (err) {
27        logger.level = 'error';
28        logger.error('query syntext error');
29         console.log(err.stack);
30       } else {
31         const jsonData = JSON.parse(JSON.stringify(result.rows));
32         console.log("jsonData", jsonData);
33
34         const csvWriter = createCsvWriter({
35           path: "csvWriter.csv",
36           header: [
37             { id: "id", title: "id"}   ,
38             { id: "name", title: "name" },
39             { id: "city", title: "city" }
40
41           ]
42         });
43         csvWriter
44           .writeRecords(jsonData)
45           .then( () =>
46           logger.level = 'info',
47           logger.info('Successfully!'),
48            console.log("csvWriter.csv successfully!")
49           );
50       }
51     });
52   });
```

here we applied a factory design pattern at Query to create an
instance for query according to the query type that the user
entered …

JS Query.js

JS QueryController.js

JS QueryFactory.js

```
5    class QueryFactory {
6
7
8            CreateQuery(queryType){
9                const log4js = require('log4js');
10               // Create the logger
11               const logger = log4js.getLogger();
12               let queryType_ = Constants.QueryType;
13               if(queryType==queryType_.READ){
14                   const read = new Query.Read();
15                   logger.level = 'info';
16                   logger.info('query viewed');
17                   return read ;
18               }
19               if(queryType==queryType_.WRITE){
20                   const write = new Query.Write();
21                   logger.level = 'info';
22                   logger.info('query written');
23                   return write ;
24               }
25               if(queryType==queryType_.INSERT){
26                   const insert = new Query.Insert();
27                   logger.level = 'info';
28                   logger.info('query iserted');
29                   return insert ;
30               }
31               if(queryType==queryType_.DELETE){
32                   const delete_ = new Query.Delete();
33                   logger.level = 'info';
34                   logger.info('query deleted');
35                   return delete_ ;
36               }
37           return null;
38       }
```

the same thing for file ..

```
class FileFactory {


        CreateFile(fileType){
            let fileType_ = Constants.FileType;
            if(fileType==fileType_.CSV){
                const csv = new File.Csv();
                return csv ;
            }
            if(fileType==fileType_.JSON){
                const json_ = new File.Json();
                return json_ ;
            }
            if(fileType==fileType_.XML){
                const xml = new File.Xml();
                return xml ;
            }
            return null;


        }

}
module.exports = FileFactory
```

and applied singleton design pattern at connection class .

```
class Connection {
    database=""
    constructor() {
        if (Connection.instance) {
            logger.level = 'info';
            logger.info('Created !');
          console.log("Instence is already created : ")
          return Connection.instance
        }
        Connection.instance = this;
        logger.level = 'info';
        logger.info('Successfully!');
        console.log("Create instence is done succssfuly : ")
    }
```

finally we using the logger in our code and print its details at
file.log :

```
 1    [2021-12-16T02:11:18.806] [INFO] default - Successfully!
 2    [2021-12-16T02:11:40.408] [INFO] default - return the host
 3    [2021-12-16T02:11:40.408] [INFO] default - return the user
 4    [2021-12-16T02:11:40.409] [INFO] default - return the port
 5    [2021-12-16T02:11:40.409] [INFO] default - return the password
 6    [2021-12-16T02:11:40.409] [INFO] default - return the database
 7    [2021-12-16T02:11:40.409] [INFO] default - return the new database
 8    [2021-12-16T23:25:59.437] [INFO] default - Successfully!
 9    [2021-12-16T23:26:18.306] [INFO] default - return the host
10    [2021-12-16T23:26:18.307] [INFO] default - return the user
11    [2021-12-16T23:26:18.307] [INFO] default - return the port
12    [2021-12-16T23:26:18.307] [INFO] default - return the password
13    [2021-12-16T23:26:18.309] [INFO] default - return the database
14    [2021-12-16T23:26:18.309] [INFO] default - return the new database
15    [2021-12-16T23:26:24.326] [INFO] default - Successfully!
16    [2021-12-16T23:26:28.125] [INFO] default - return the host
17    [2021-12-16T23:26:28.126] [INFO] default - return the user
18    [2021-12-16T23:26:28.126] [INFO] default - return the port
19    [2021-12-16T23:26:28.126] [INFO] default - return the password
20    [2021-12-16T23:26:28.126] [INFO] default - return the database
21    [2021-12-16T23:26:28.127] [INFO] default - return the new database
22    [2021-12-16T23:30:03.231] [DEBUG] default - query  controller enter!
23    [2021-12-16T23:30:03.231] [INFO] default - query viewed
24    [2021-12-16T23:30:03.231] [INFO] default - query return
25    [2021-12-16T23:30:16.885] [INFO] default - show the rows
26
```

and we using exceptions in our code ::
you will see more examples at the code ..

```
    catch(e){
      logger.level = 'error';
      logger.error('The id is not available...');
      console.log("Authentication failed, try again..")

      Implementation(client);
    }
```

THE END :)