

Module: Data structure & algorithms 2.

The Report

Mini-project: Electricity Company Management system.

Team members:

1. Djamel BOUGHEDDOU
2. Bilal FELLAH
3. Zyad KHERRAF
4. Hamou DJELLAB

● **Introduction**

The world today is witnessing a very rapid transformation in the **energy** field, away from fossil fuels and toward greener sources of power. Not only is it a necessary step toward environmental sustainability, but also an interesting challenge and opportunity for electricity firms. For example, if a customer can be an individual who **generates electricity** with solar photovoltaics and supplies it to the network, then consumers as a whole become **prosumers** (a portmanteau term blending ‘producer’ and ‘consumer’).

This transition holds great promise for **Algeria**. Statistics from **International Energy** indicates that in 2021, Algeria produced about **77 billion kilowatt-hours of electricity**. Algerians consume, on average **1.484 kWh per capita annually**, as stated by the **World Bank**. According to **Statista**, in 2021, natural resources were plentiful in the country but renewable energy sources had yet to be tapped.

This report describes the development and structure of Algeria ’ s national network-based **Electricity Company Management System (ECMS)**, which governs power consumption, billing, and energy production by customers. Its objectives encompass not only optimizing energy consumption, but also creating a culture of renewable energy among Algeria's people.

Abstract:

This report will tackle every detail done in this mini-project ECMS.

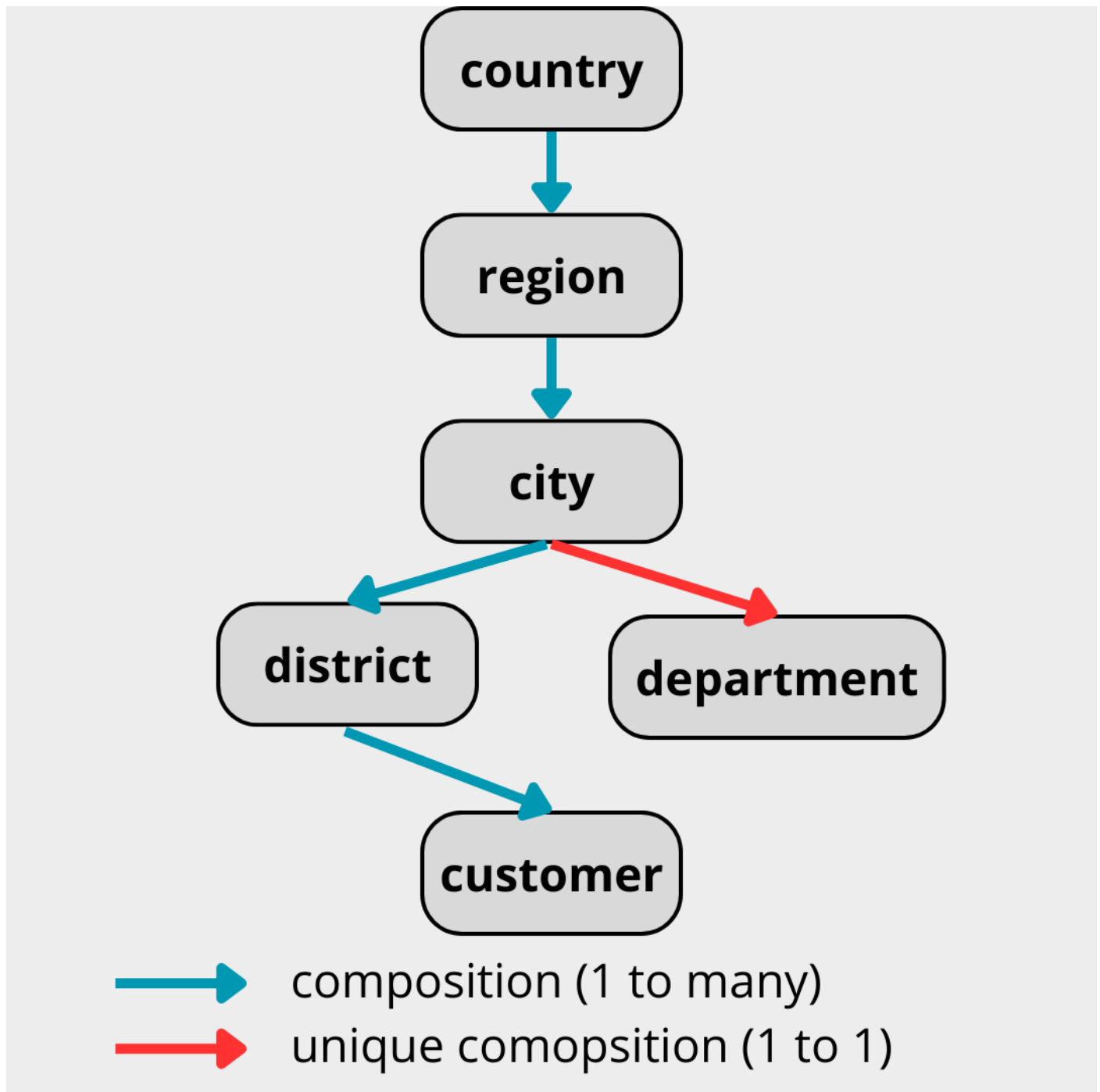
The solution design was first made by hash tables (`unordered_map` from STL) in order to store the household data (i.e. customer: it has a name, ID number of the electricity account holder which is **unique...**), we will see more details in the **code architecture** section about how all the ADT are implemented. The program also was done using AVL tree and Binary Search tree in order to **sort** the **customers** based on the amount of injection for the prize and to **sort** the **departments** from best performing to the least performing based on the amount paid by the customers of the city they belong.

Sections:

- Code Architecture
- Dataset details
- Guideline and Features
- Performance and Complexity analysis
- Conclusion
- Appendix

• Code Architecture:

We made several classes and ADTs in order to have a control on ECMS and the below diagram illustrates the structure



• Structure details:

As explained in the diagram, the classes are nested in that order. The **country**, **region** and **city** classes are **similar** in their **structure**, they have a **unique name** and a **vector** of their **composed** classes (ex: a country has a vector of regions). We have chosen a vector because we know that the number of regions is small (finite) and even looping over all the vector will be $O(1)$. This structure goes down to **city class** when it has a vector of districts and **only one department** object (each city has its own department)

Now for the **district class**, it has some another ADT which is **unordered_map** from STL, since each customer (i.e. household) can be identified uniquely by “ID number of electricity account”, we could use **hashing** (The id as a **Key**) to get the best time complexity for search, insert, delete (more details in ‘performance and complexity analysis’)

Here are some figures:

```
class Country{
private:
    string name;
    vector<Region> Regions;

public:
    Country(const string& = "no-name", const vector<Region>& = {});
    Country(string&&, vector<Region>&&);

    Country(const Country&);
    Country(Country&&);

    Country& operator=(const Country&);
    Country& operator=(Country&&) noexcept;

    //Setters
    void setName(const string&);
    void setName(string&&);

    void setAllRegions(const vector<Region>&);
    void setAllRegions(vector<Region>&&);

    //Getters
    const string& getName() const;
    const vector<Region>& GetAllRegions() const;

    //add new customer depending on its address
    bool addCustomer(const Customer&); //returns if done successfully
};
```

```
class Region {
private:
    string regionName;
    vector<City> cities;

public:
    Region(const string& = "no-name", const vector<City>& = {});
    Region(string&&, vector<City>&&);

    Region(const Region&);
    Region(Region&&) noexcept;

    Region& operator=(const Region&);
    Region& operator=(Region&&) noexcept;

    void setName(const string&);
    void setName(string&&);

    void addCity(const City&);
    void addCity(City&&);

    void addAllCities(const vector<City>&);
    void addAllCities(vector<City>&&);

    const string& getName() const;
    const vector<City>& getAllCities() const;

    bool addCustomer(const Customer&);
};
```

```
class City {
private:
    string cityName;
    Department city_dep;
    vector<District> districts;

public:
    City(const string& = "no-name", const Department& = {}, const vector<District>& = {});
    City(string&&, Department&&, vector<District>&&);
    City(const City&);
    City(City&&) noexcept;
    City& operator=(const City&);
    City& operator=(City&&) noexcept;

    void setName(const string&);
    void setName(string&&);

    void set_city_dep(const Department&);
    void set_city_dep(Department&&);

    void addDistrict(const District&);
    void addDistrict(District&&);

    void setAllDistricts(const vector<District>&);
    void setAllDistricts(vector<District>&&);

    const string& getName() const;
    const Department& getDepartment() const;
    const vector<District>& getAllDistricts() const;

    bool delete_district(const District&);

    bool addCustomer(const Customer&);
};
```

Country, region and city classes follow the same logic of structure

```
class District {
private:
    string districtName;
    unordered_map<unsigned int, Customer> customers;

public:
    District(const string& = "no-name", const vector<Customer>& = {});
    District(string&&, vector<Customer>&&);
    District(const District&);
    District(District&&) noexcept;
    District& operator=(const District&);
    District& operator=(District&&) noexcept;

    void setName(const string&);
    void setName(string&&);
    void addCustomer(const Customer&);
    void addCustomer(Customer&&);

    const string& getName() const;
    const unordered_map<unsigned int, Customer>& getAllCustomers() const;
    const Customer& getCustomer(int id) const;
};
```

The district class uses unordered_map in order well organize the customers

```

class Customer
{
private:
    string name;
    string address;
    vector<int> family_ages;           //we can extract the number of members from the vector size
    unsigned int ID_Num_E_acc;        //should be unique (primary key)
    double consumption, production;   // (Kilowatt/hours)
public:
    //scratch constructors with default values
    Customer(const string &Name = "no-name", int ID = 0, const vector<int>& f_age = {}, const string& address = "no-adrs");
    Customer(string &&name, int ID, vector<int>&& f_age, string&& address);
    //copy constructor
    Customer(const Customer&);
    //move constructor
    Customer(Customer&&);
    //copy assignment
    Customer& operator= (const Customer&);
    //move assignment
    Customer& operator= (Customer&&);
    //distractor
    ~Customer();
    //insertion and extraction operators:
    friend ostream& operator << (ostream& , const Customer&);
    friend istream& operator >> (istream& , Customer&);
    //overloading operators (needed)
    bool operator!= (const Customer&);
    bool operator== (const Customer&);
    bool operator< (const Customer&);
    bool operator<= (const Customer&);
    bool operator> (const Customer&);
    bool operator>= (const Customer&);
    Customer operator+ (const Customer&);
    //Setters
    Customer& set_name(const string&); //for cascading
    Customer& set_name(string&&);
    Customer& set_family_ages(const vector<int>&);
    Customer& set_family_ages(vector<int>&&);
    Customer& set_address(const string&);
    Customer& set_address(string&&);
    Customer& set_consumption(double);
    Customer& set_production(double);
    //Getters
    const string& get_name() const;
    string& get_name();
    const vector<int>& get_family_ages() const;
    vector<int>& get_family_ages();
    int get_family_num() const;
    const string& get_address() const;
    string& get_address();
    unsigned int get_ID() const;
    double get_consumption() const;
    double get_production() const;
};

```

The customer and department classes have overloaded operators in order to sort them in AVL and BS tree

```

class Departement
{
private:
    double budget, TotalAmountCustomer, monthlySpent;
public:
    //no need for move constructor
    Departement(double = 0, double = 0, double = 0);
    Departement(const Departement& );

    // the assignment operator
    Departement& operator=(const Departement& );

    // the comparison operators
    bool operator>(const Departement& ) const;
    bool operator>=(const Departement& ) const;
    bool operator<(const Departement& ) const;
    bool operator<=(const Departement& ) const;
    bool operator==(const Departement& ) const;
    bool operator!=(const Departement& ) const;
    // addition operator
    Departement operator+(const Departement& dep) const;

    // printing the data of a Departement
    friend ostream& operator<<(ostream& read, const Departement& Departement);
    // getting the data of a Departement
    friend istream& operator>>(istream& write , Departement& Departement);

    // getters
    double getBudget()const;
    double getTotalAmountCustomer()const;
    double getMonthlySpent()const;

    // setters
    void setBudget(double);
    void setTotalAmountCustomer(double);
    void setMonthlySpent(double);
};

```

• Utility classes:

Since the date and the address are very important in our structure, we made an input validation for both; date_validation, and address_validation:

```
#ifndef ADDRESS_VALIDATION_H
#define ADDRESS_VALIDATION_H
#include <string>
#include <vector>

using namespace std;

bool isValidAddressFormat(const string& address);
string removeSpaces(const string& str);
vector<string> splitAddress(const string& address);
vector<string> processAddress();

#endif
```

The implementation uses <regex>
<sstream> <iomanip>
source of regex :
<https://www.geeksforgeeks.org/regex-library-c-stl/>

The implementation uses <chrono>
<sstream> <iomanip>
source of chrono :
<https://www.geeksforgeeks.org/date-and-time-parsing-in-cpp/>

```
#ifndef DATE_VALIDATION_H
#define DATE_VALIDATION_H

#include<string>
using std::string;

bool isValidDateFormat(const string& input);
string getUserInputDate(const string& prompt);
bool isStartDateBeforeEndDate(const string& startDate, const string& endDate);
string incrementDate(const string& inputDate);
string incrementDateByOneMonth(const string& inputDate);
string incrementDateByOneYear(const string& inputDate);
void processDates(string& startDate, string& endDate);

#endif
```

• Dataset details:

1.1 Data Organisation Structure:

Single-location Storage:

The dataset is stored in a centralized location for streamlined access and management.

Hierarchical Folder Structure:

Geographical Hierarchy:

Organized from country to region, city, and district, reflecting Algeria's geographical layout. The dataset encompasses nine regions within Algeria. Each region comprises between 3 and 11 cities, representing Algerian wilayas. In each city, data is collected from three districts (dairas)(some cities like Djanet has only one or two districts). Within each district, there are 500 customers, making a total of 1,500 customers per city in general, and 82500 customers in total.

Customer Data Arrangement:

Districts:

Contain 366 daily JSON files named as yyyy-mm-dd.json, conveniently enabling efficient access and manipulation of daily data for the year 2020.

Within city folders:

One file is dedicated to department data. A yearly weather file captures weather details for the entire year. Three folders correspond to different districts within the city.

1.2 Advantages of the Organizational Structure:

The hierarchical structure mirrors real-world relationships between entities, offering natural organization and efficient access patterns. Supports easy aggregation and summarization of data at various levels (district, city, region) for analytical purposes.

Ease of Access: Utilizing the yyyy-mm-dd.json naming convention for daily files simplifies access and manipulation of specific datasets, enhancing overall efficiency. All necessary datasets can be accessed simultaneously, with minimal navigation required.

Scalability: The structured layout supports seamless data scaling without compromising access or manageability.

Improved Search and Retrieval: Logical organization ensures swift and precise data searches, enhancing overall efficiency in data retrieval tasks.

Consistency Across Departments: Adhering to a standardized structure minimizes confusion, encourages collaboration, and maintains uniformity across departments.

2.1 File Types and Formats:

Daily Customer Data:

File Format: JSON files named in the format yyyy-mm-dd.json.

Content: 500 customers with detailed information including consumption, production, ID, number and ages of family members, and address. The address is of the form region\city\district, we could obtain these 3 data points about the customer directly from the address.

Departmental Data:

File Format: JSON files named in the format city_department.json, capturing monthly expenditure from the allocated budget.

Content: Month and the amount spent from the budget in this month.

Weather Files:

File Format: JSON files named in the format city.json, containing weather details for each day of the year 2020.

Content: Month, day, the weather conditions (rainy, sunny, cloudy, snowy), maximum and minimum temperatures for each day, and the count of sunny hours.

2.2 File Type Selection:

Choice of File Format: JSON (JavaScript Object Notation).

Human-Readable Format: JSON is easy to read and understand, facilitating manual inspection and debugging.

Suits Hierarchical Data: JSON aligns well with the tree-like data structure used, allowing nested representation of hierarchical data.

Efficient Size: JSON files are lightweight compared to XML or CSV formats. They consist of key-value pairs enclosed by curly brackets.

Interoperability: Widely supported across different programming languages and platforms, ensuring flexibility in data exchange and integration.

Semi-structured Data: Allows flexibility in adding or removing fields/attributes without strict schema enforcement, accommodating potential future data changes.

2.3 Advantages of JSON Files:

Rich Structure: JSON accommodates complex nested data structures, offering more descriptive data representation compared to flat formats like CSV.

Simplicity and Readability: Human-readable format simplifies understanding and inspection, aiding in data validation and manipulation.

Flexibility: Accommodates potential changes in data structure without strict schema enforcement, supporting adaptability to future data modifications.

3.Data :

All the data for the project was generated randomly by implementing a specific function for that, respecting some logical, statistical, and realistic standards. That is because there is no source for real data. (We tried to contact a source, but we did not manage to get the data.)

We decided to take regions as the name means: “Regions”, so we divided Algeria to 9 regions, center, north and south, then in each of the three there are the central, eastern and western regions.

This choice aims to simplify the work when dealing with customers: if we define a region as a wilaya, the number of districts will be 1541, which is very complicated to handle. Also, this choice organizes our work and makes it clear for any user.

This implies that the cities will be the 58 wilayas, and the districts will be the Dairas, which is logical since the daïra is a district.

We took a sample from each city (1500 customers) to cover the whole country.

A set of Algerian names and surnames was the source used to generate the customers' names randomly. (The full names could be real, but they do not have to.)

According to Statista.com, the Algerian consumption per capita in 2020 was around 1.6 mwh, which makes it around 4.5 kwh per day, and that is what we took into consideration when generating data to make it more realistic.

Obviously, there is no data about individual production in Algeria since there is no individual production in Algeria. That is what makes generating this data a bit harder. We assume that a customer can produce 0 Kwh in a day and make the upper bound here less than the one for consumption.

The customer ID was random but unique at the same time; a function "getUniqueID()" was implemented for that. The ID is a unique number between 10000000 and 99999999 since the number of customers cannot pass these values.

File names, folder structures, and data organization were crafted to reflect a logical, hierarchical organization mirroring the geographic layout and data categorization for ease of access and retrieval.

● Guideline and Features:

● Guideline:

1. In the 'MAIN_folder', open the 'main.cpp'
2. There are **two main functions** (first one is meant for mining data and the second one for retrieve the winner prize)
3. Follow the instructions and the comments provided in the 'main.cpp' to get the best results

● Features:

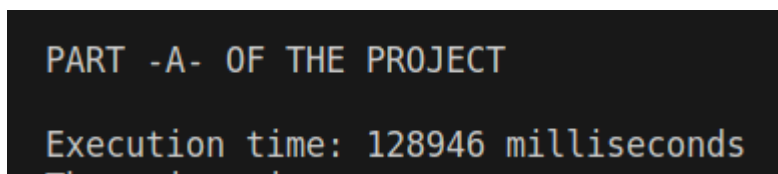
- The powerful structure of storing data decreases insertion/deletion/search time dramatically
- Display the bill of any customer (in our data set) any date (you could specify the starting date and ending date)
- Display the bill of many customers at ones (also you could specify the range)
- See the performance of each department (sorted)
- See details about the weather, temperature and number of sunny hours in any day (the distribution of values is realistic !!)

● Performance and Complexity analysis:

1. PART A -BST-:

We have included the **BinarySearchTree** template class provided in the lecture, we run the demo and selected all the queries and got these results :

(demo of finding a national winner prize for January 2020)



```
PART -A- OF THE PROJECT
Execution time: 128946 milliseconds
```

This time includes the time of reading data from the files (Our dataset is so huge that's way it takes approximately 13 seconds)

We have done all the required test and are ready in the 'main.cpp'

2. PART B -AVL tree-:

We have included the **AvlTree** template class provided in the lecture, we run the demo and selected all the queries and got these results :

(demo of finding a national winner prize for January 2020)

```
PART -B- OF THE PROJECT
Execution time: 115176 milliseconds
The prize winner
```

As we can see here, there is a difference in time execution (AVL tree performs better than BST)

Also you can find all the test in the '**main.cpp**' (finding best department sorted from best to worst, display the bill of a specific customer, display the bills of all customers ...)

• Conclusion:

All in all, the introduction of an ECMS signifies a step toward rationalizing energy usage and production throughout Algeria. If the ECMS can manage power consumption, calculate bills, and get customers to produce energy themselves, then this would change Algeria's entire energy landscape.

A more flexible and human-friendly system that lets consumers become prosumers not only cuts back on energy waste but also creates opportunities for a greener, cleaner future. Although some difficulties have cropped up in the implementation of the ECMS, the benefits it offers underscore its importance in the context of Algeria's energy sector.

The work that remains to be done on the ECMS could include better features, a more enjoyable user experience, and coverage for more regions. Moving forward, the ECMS will surely play an important part in shaping Algeria's energy destiny.

As Thomas Edison once said, "We are like tenant farmers chopping down the fence around our house for fuel when we should be using Nature's inexhaustible sources of energy--sun, wind and tide. I'd put my money on the sun and solar energy. What a source of power! I hope we don't have to wait until oil and coal run out before we tackle that." This quote underscores the importance of harnessing renewable energy sources, a goal that the ECMS is designed to promote.

• Appendix:

1.Tasks division:

All the team members helped in:

- .Solution design idea.
- .Storing data files structure.
- .The algorithms of most of the system functions.
- .Verify, debug and test the code.

Djamel BOUGHEDDOU:

Classes creation:

Created the class "Customer"

As shown in the figure above, (big five, data members, operations overloading...)

Data input validation Folder:

Date validation

Address validation

both includes many utility functions (details in source code)

<https://www.geeksforgeeks.org/regex-library-c-stl/>

<https://www.geeksforgeeks.org/date-and-time-parsing-in-cpp/>

MAIN_folder :

bill_func :

It synchronizes between interface and dataset

all the needed functions (customer bill, best departments ...)

This file relies on 'file_config' folder

Find_prize_winner :

It has many functions on it to perform the task of finding the winner prize (works with <filesystem> ::is_directory, details in the source code) (<https://www.geeksforgeeks.org/file-system-library-in-cpp-17/>)

main.cpp :

The interface of the program, includes all the required queries (has two main functions, each on handles a part of the project, details are in the comment section in the source code)

BILAL FELLAH:

Classes creation:

Created the class "Department".

Declared all the needed data members, implemented the big five , the setters and getters, and the overloaded operators needed.

File Handling and configuration:

. Designed the functions to read/write data from/into the files:

Customer data functions:

- . a function that reads the information of one customer according to its id.
- . a function to get the information about all the customers of a specific district.
- . a function to add a new customer to the files in a specific path and a month.

Department data functions:

- . a function to read the monthly spent amount from the budget of a given department.

. a function that adds the information about a department to the data files.

Djellab Hamou:

Classes creation:

Created the class "Country".

Declared all the needed data members, implemented the big five , the setters and getters, and the overloaded operators needed.

Data Generation and Modeling:

Gather information about the Administrative division in Algeria to use for simulating the management system according to the country's administrative division.

- Divided Algeria into 9 regions, distributed cities and districts using Wikipedia

https://ar.wikipedia.org/wiki/%D9%88%D9%84%D8%A7%D9%8A%D8%A7%D8%AA_%D8%A7%D9%84%D8%AC%D8%B2%D8%A7%D8%A6%D8%B1.

-Worked in generating customers' data, proposing many solutions, functions and ideas that lead to the final function.

-Sorted the data based on the hierarchy mentioned above in Json files..

-Generated the data of the weather , , before that searched about the climates in Algeria :

<https://www.climatestotravel.com/climate/algeria/algiers>

Zyad Kherraf:

Classes creation:

Created the classes "Region", "City" and "District".

Declared all the needed data members, implemented the big five , the setters and getters for each class, and the overloaded operators needed..

Data Generation and Modeling:

Created functions to generate synthetic data mimicking real-world energy consumption and production patterns in Algeria and marketing departments outcomes.

Modeled customer data based on statistical averages, incorporating parameters like ID, energy consumption, production, and family details.

Utilized specific algorithms and logical standards to ensure generated data remained plausible and realistic within Algerian standards.

Generated data for 82500 customers and 58 departments, sorted them in daily (yearly for the department)JSON files, sorted the data following the hierarchy explained before.

Documentation and Reporting:

Compiled comprehensive reports outlining the dataset's structure, file types, and data generation methods.

Documented the introduction of the report, the dataset part and the appendix.

2.Functions to generate data:

All the data were generated using:

C++ functions implemented by Zyad Kherraf, to generate customers' data and departments' data.

C++function implemented by Hamou Jellab to generate weather's data.

Attached is the link for these functions code and the dataset:

<https://drive.google.com/drive/folders/1PHOKVexrhkov8LbOZ719TIZVMnuwkPVE?usp=sharing>