UNIVERSITY OF IOANNINA

DEPARTMENT OF C.S.E.

ADVANCED TOPICS IN DATABASE TECHNOLOGY AND APPLICATIONS

PROJECT FOR ACADEMIC YEAR 2022-2023

KARATHANOS GEORGIOS, 4691

SYRROS STYLIANOS, 4805

REPORT

MAY 2023

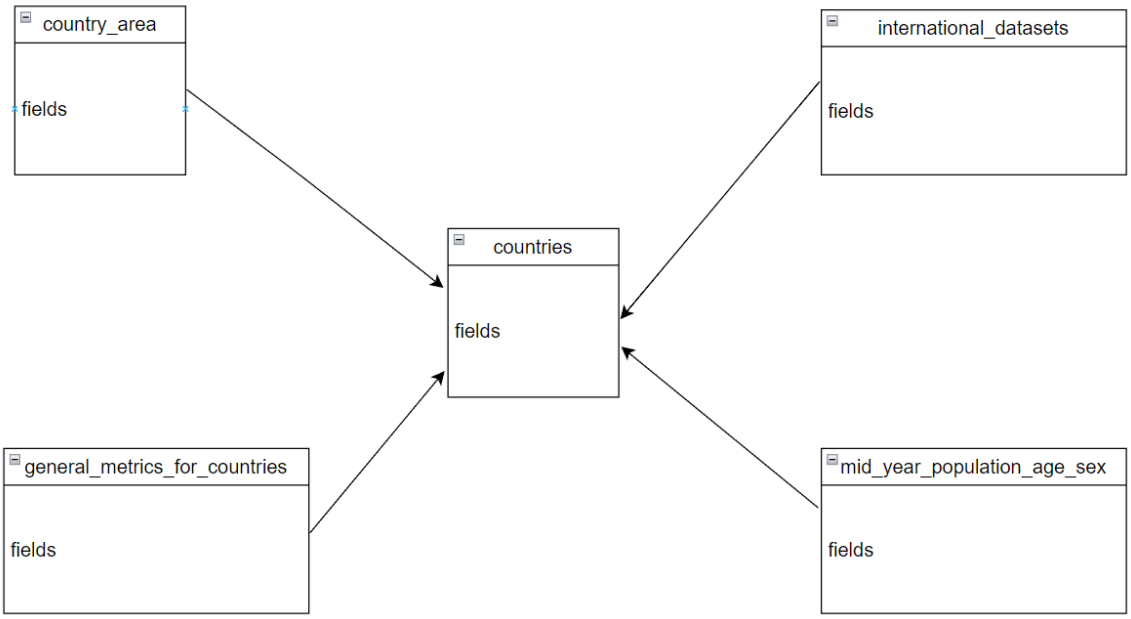
HISTORY OF PREVIOUS VERSIONS

Date	Version	Description	Writer
2023/03/31	1.0	Load of income-data	George Karathanos
2023/04/01	1.1	Load of age-data	Stylianos Syrros
2023/04/02	1.2	Final changes to the database schema and backup	George Karathanos & Stylianos Syrros
2023/05/30	2.0	The rest of this report	George Karathanos & Stylianos Syrros

1 DATABASE

In this chapter, we present the database schema and how it was created in MySQL.

1.1 DATABASE



Picture 1.1 Database schema

```sql
CREATE TABLE COUNTRIES (
  ISO CHAR(2),
  ISO3 CHAR(3),
  ISO_Code INT PRIMARY KEY,
  FIPS CHAR(2),
  Display_Name VARCHAR(50),
  Official_Name VARCHAR(100),
  Capital VARCHAR(50),
  Continent VARCHAR(50),
  CurrencyCode CHAR(3),
  CurrencyName VARCHAR(50),
  Phone VARCHAR(70),
  Region_Code INT,
  Region_Name VARCHAR(50),
  Sub_Region_Code VARCHAR(10),
  Sub_Region_Name VARCHAR(50),
  Intermediate_Region_Code INT,
  Intermediate_Region_Name VARCHAR(50),
  Status VARCHAR(70),
  Developed_or_Developing VARCHAR(50),
  SIDS VARCHAR(10),
  LLDC VARCHAR(10),
  LDC VARCHAR(10),
  Area_SqKm INT,
  Population INT
);
```

```sql
CREATE TABLE IF NOT EXISTS COUNTRY_AREA (
  ISO_Code INT,
  country_code CHAR(2),
  country_name VARCHAR(50),
  country_area REAL,
  PRIMARY KEY (ISO_Code),
  FOREIGN KEY (ISO_Code) REFERENCES COUNTRIES(ISO_Code)
);
```

```sql
CREATE TABLE IF NOT EXISTS INTERNATIONAL_DATASETS (
    ISO_Code INT,
    year INT,
    country_code CHAR(2),
    country_name VARCHAR(50),
    fertility_rate_15_19 FLOAT,
    fertility_rate_20_24 FLOAT,
    fertility_rate_25_29 FLOAT,
    fertility_rate_30_34 FLOAT,
    fertility_rate_35_39 FLOAT,
    fertility_rate_40_44 FLOAT,
    fertility_rate_45_49 FLOAT,
    total_fertility_rate REAL,
    gross_reproduction_rate REAL,
    sex_ratio_at_birth REAL,
    crude_birth_rate REAL,
    crude_death_rate REAL,
    net_migration REAL,
    rate_natural_increase REAL,
    growth_rate REAL,
    midyear_population INT,
    total_flag VARCHAR(1),
    starting_age INT,
    age_group_indicator VARCHAR(1),
    ending_age INT,
    midyear_population_sex INT,
    midyear_population_male INT,
    midyear_population_female INT,
    infant_mortality REAL,
    infant_mortality_male REAL,
    infant_mortality_female REAL,
    life_expectancy REAL,
    life_expectancy_male REAL,
    life_expectancy_female REAL,
    mortality_rate_under5 REAL,
    mortality_rate_under5_male REAL,
    mortality_rate_under5_female REAL,
    mortality_rate_1to4 REAL,
    mortality_rate_1to4_male REAL,
    mortality_rate_1to4_female REAL,
    PRIMARY KEY (ISO_Code, year),
    FOREIGN KEY (ISO_Code) REFERENCES COUNTRIES(ISO_Code)
);
```

```sql
CREATE TABLE IF NOT EXISTS GENERAL_METRICS_FOR_COUNTRIES (
  ISO_Code INT,
  Year INT,
  Country_Name VARCHAR(100),
  Income_Index REAL,
  Labour_Share_GDP REAL,
  Gross_fixed_capital_formation REAL,
  GDP_Total REAL,
  GDP_Per_Capita REAL,
  GNI_Per_Capita INT,
  Estimated_GNI_Male INT,
  Estimated_GNI_Female INT,
  Domestic_Credits REAL,
  PRIMARY KEY(ISO_Code, YEAR),
  FOREIGN KEY (ISO_Code) REFERENCES COUNTRIES(ISO_Code)
);

```

```sql
CREATE TABLE IF NOT EXISTS Midyear_Population_Age_Sex (
  ISO_Code INT,
  country_code CHAR(2),
  country_name VARCHAR(50),
  year INT,
  sex VARCHAR(10),
  max_age INT,
  population_age_0 INT,
  population_age_1 INT,
  population_age_2 INT,
  population_age_3 INT,
  population_age_4 INT,
  population_age_5 INT,
  population_age_6 INT,
```

```sql
  population_age_90 INT,
  population_age_91 INT,
  population_age_92 INT,
  population_age_93 INT,
  population_age_94 INT,
  population_age_95 INT,
  population_age_96 INT,
  population_age_97 INT,
  population_age_98 INT,
  population_age_99 INT,
  population_age_100 INT,
  PRIMARY KEY (ISO_Code,year,sex),
  FOREIGN KEY (ISO_Code) REFERENCES COUNTRIES(ISO_Code)
);
```

1.2 DATABASE SCHEMA IN PHYSICAL LEVEL

1.2.1 PARAMETERS OF DBMS

We didn't change any of the DBMS configuration settings, so we worked with the default MySQL values which are INNODB for the storage engine and 128Mb buffer pool size

```
# The default storage engine that will be used when create new tables when
default-storage-engine=INNODB
```

```
# a busy server.
innodb_buffer_pool_size=128M
```

1.2.2 PHYSICAL LEVEL SETTINGS

In our database we chose to use five tables for the proper separation of data and to make it easier to retrieve the information we need.
1) The first table is the countries table which contains information for each country in our database such as ISO_Code , Display_Name, Capital and others
2) The second table is country_area which contains information about the area of each country
3) The third is general_metrics_for_countries and it mainly contains information about various metrics related to the economy of each country , such as Income_Index and GDP_Total .
4) The fourth table is international_datasets which contains for each country information on demographics and population data for various countries , such as total_fertility_rate and gross_reproduction_rate
5)The fifth and last table is the mid_year_population_age_sex table which contains information on population and age variation for different countries.

1.2.3 SECURITY & BACKUP

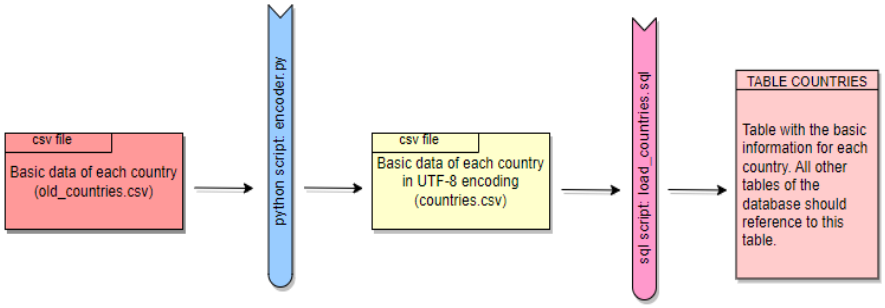
Regarding the security settings for the base we have taken care to keep in a separate part of our computer backup files with which we can rebuild the base from scratch in case it is damaged. The backup files contain the csv files with the necessary database data , the filters (python scripts) that we applied to the csv files to bring them in the format we want and the sql scripts with which we built our database .

2 SOFTWARE ARCHITECTURE

2.1 ETL

First, we should highlight the fact that the final metrics that we want to visualize about countries can be categorized into 2 main categories: Data that has to do with income, and data that has to with metrics based on age. Therefore, we grouped the raw data into 2 groups and managed them separately.

Before we explain the processing of that data, we should declare a table in our database that will represent the countries we study, in which there will be general information about them (like the ISO and Capital). In order to achieve that, instead of loading the data straight to our MySQL database, we first pass them from a filter that transforms the csv file that holds them to UTF-8 encoder for parsing reasons.



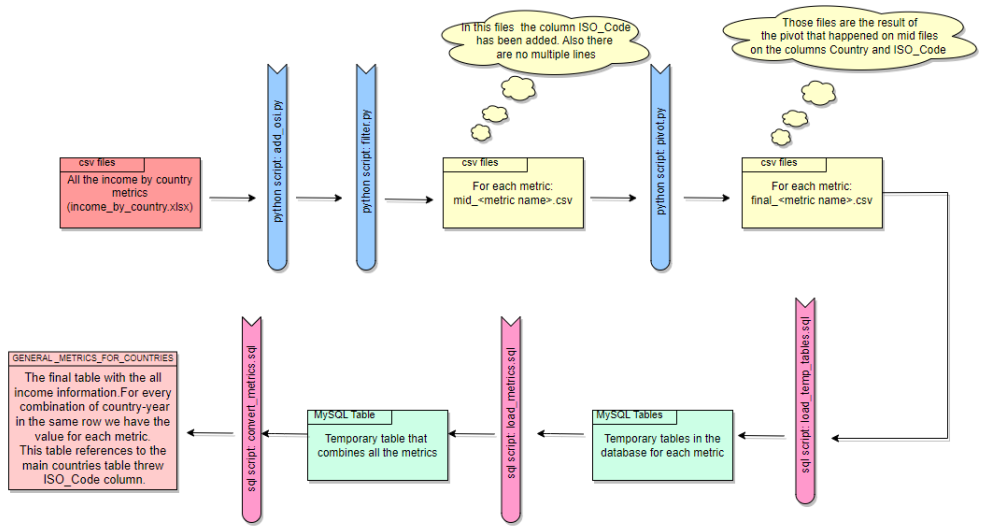
Picture 2.1 Country data processing pipeline

Below there are the pipelines for the processing of the 2 groups of data we mentioned.

2.1.1 INCOME DATA

Our input data are 9 csv files (into one xlsx as sheets). Consider the python scripts as filters that achieve a transformation to the structure of the data. Firstly, with 2 scripts we add a new column to every csv that represents the ISO Code of each country and in this way, we will later achieve a mapping with the basic country table we referenced earlier. Also, we throw duplicate lines we found and save the result to new csv files starting with prefix 'mid_'. After that, we have a next filter layer that is a python script that implements a pivot to all the csv files on the columns 'Country'. As a result, we have a csv file for each metric which starts with the prefix 'final_' and for columns has the country (with the ISO Code) they year and the metric we measure for the specific combination of country-year. As we noticed, in the raw data some boxes where empty and as a result in the final files some metrics have null values, so we pass the final files from a filter that eliminates these dead rows which is the null_filter.py script and it is given in the source code of the application with the other scripts *(this filter is not appeared in the following diagram as it does not have a new output but refresh the existing final files)*. After that we have our data ready to be loaded in separate tables for each metric and we do that with the script load_temp_tables.sql which fills the metrics tables with the values of the final files. As we discussed in the previous chapter, we want all our metrics in the same table for faster querying performance so with the load_metrics.sql

script we combine all the tables into one that has all the metrics as columns and for primary key has the combination of ISO Code and Year. The last thing we had to do was to join this table with the basic countries table in order to add to this table the official name for each country in order to have a proper mapping. After that we have our GENERAL_METRICS_FOR_COUNTRIES table ready and set in our database. Below there is the pipeline of this process.

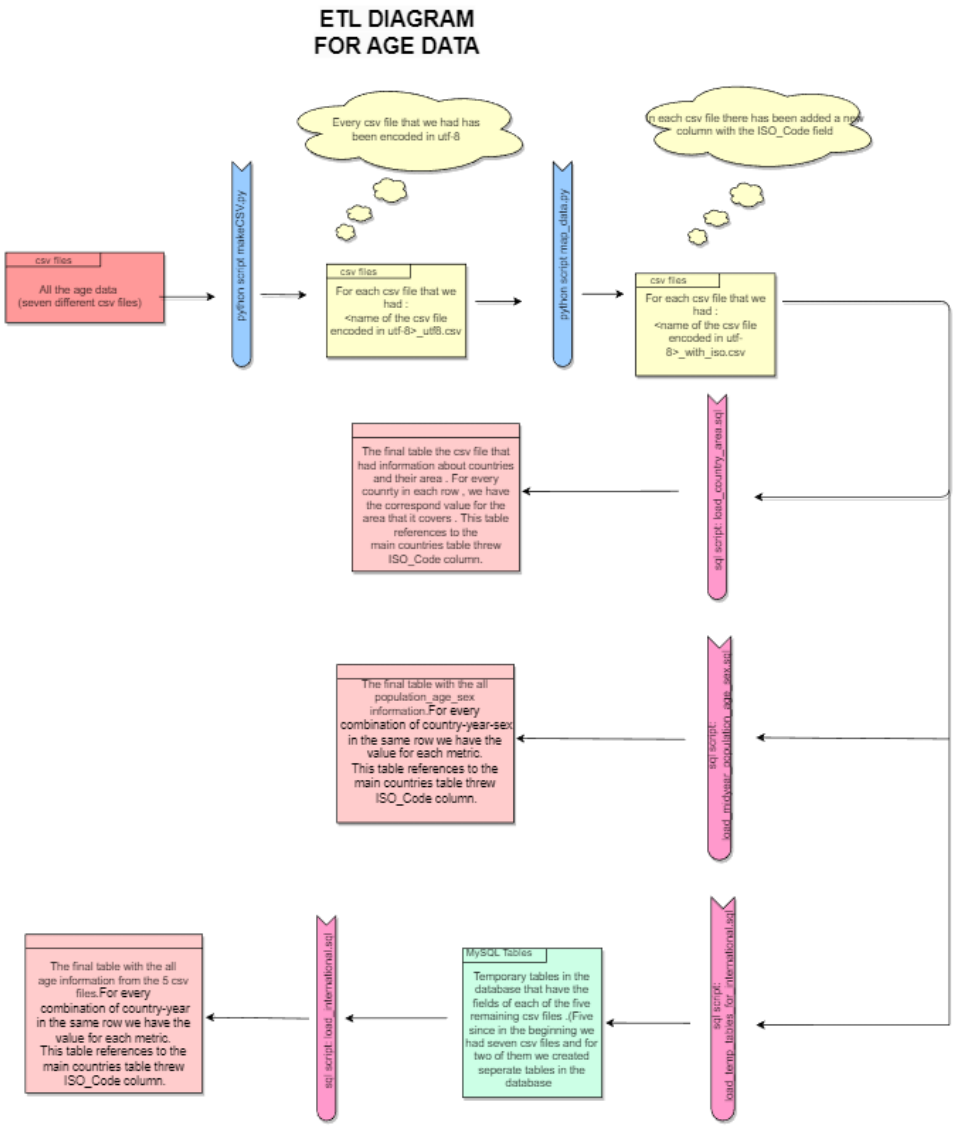


Picture 2.2 Income data processing pipeline

2.1.2 AGE DATA

Our input data are 7 csv files . Consider the python scripts as filters that achieve a transformation to the structure of the data . We used two scripts for the age data . The first one was the makeCSV.py script which encodes each of the seven CSV files containing age data for countries in the utf-8 format and saved the results to new CSV files with the suffix 'utf-8'. The second one was the map_data.py script that maps the data from the 'ISO_Code' field to the data from the 'FIPS' field in the 'countries.csv' file with the help of the iso_to_fips dictionary that we created . Afterwards , in the same script we add a new column to every csv that represents the ISO Code of each country and in this way, we will later achieve a mapping with the basic country table we referenced earlier. Also, we throw all the duplicate lines that we found and save the results to new csv files with suffix '_with_iso' . In this case , we didn't need another filter layer(python script) to implement a pivot to the csv files on the columns 'Country' since the csv file are already in the desired format , which means they have the 'ISO_Code' field and after that the other metrics . At that the point , we had our data ready to be loaded in separate tables depending on the fields of each file . We created three different tables for the seven CSV files that we had . The first table that we created was the 'country_area' table for the 'country_names_area.csv' file . This file had only one common field with the othe CSV files , which was the 'ISO_Code' , and it served as the primary key for the table . In contrast, the other files had both the 'ISO_Code' and 'year' fields in common . We then loaded the data to the 'country_area' table with the load_country_area.sql script . The second table that we created was the 'midyear_population_age_sex' table for the 'midyear_population_age_sex.csv' file . We decided to make a separate table for this file since its primary key was the combination of 'ISO_Code' , 'year' and 'sex' and not just the combination of 'ISO_Code' and 'year' like the other CSV files . We then loaded the data to

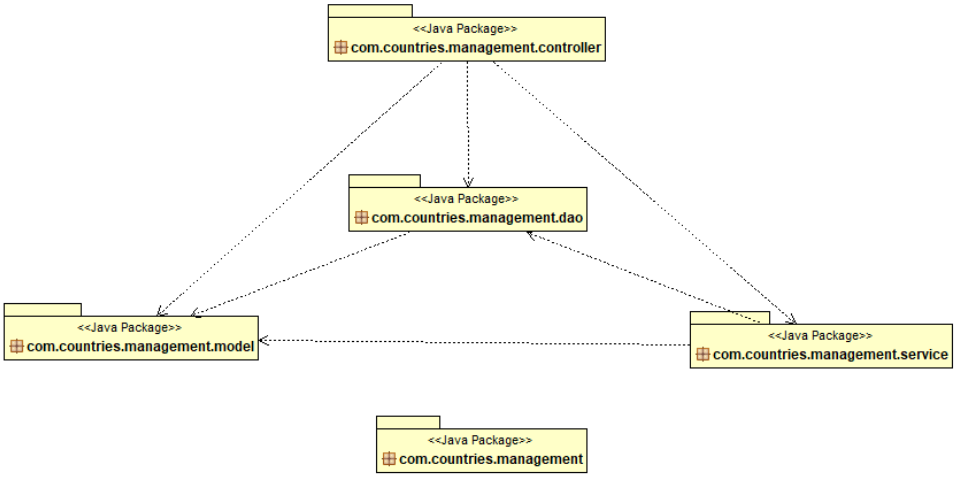
the ‘midyear_population_age_sex’ table with the load_midyear_population_age_sex.sql script . At this point we created temp tables for each of the five remaining CSV files and loaded the data with the script load_temp_tables_for_international.sql which fills the metrics tables with the values of the final files . Finally , the third and last table that we created was the ‘international_datasets’ table which is the table for the remaining five CSV files that we created the temp tables for . The primary key of this table is the combination of ‘ISO_Code’ and ‘year’ and it consists of all the fields of the five CSV files . After loading the data in this table with the load_international_datasets.sql script we have all of our three tables **international_datasets , country_area** and **midyear_population_age_sex** ready and set in our database. Below there is the pipeline of this process.



Picture 2.3 Age data processing pipeline

2.2 PACKAGE DIAGRAM

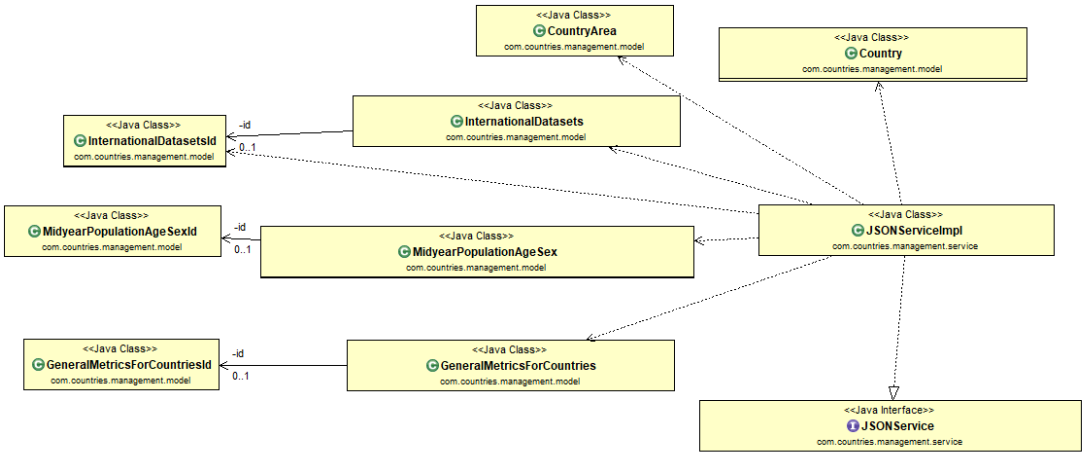
Below there is the package diagram of the spring application. It is obvious that the application was based on the MVC pattern.



Picture 2.3 Package diagram

2.3 CLASS DIAGRAMS

Below there is the class diagram of the spring application. The fields and the methods of each class are not visible because this would make hard for the reader to understand the dependencies between the classes which is the point of the diagram.



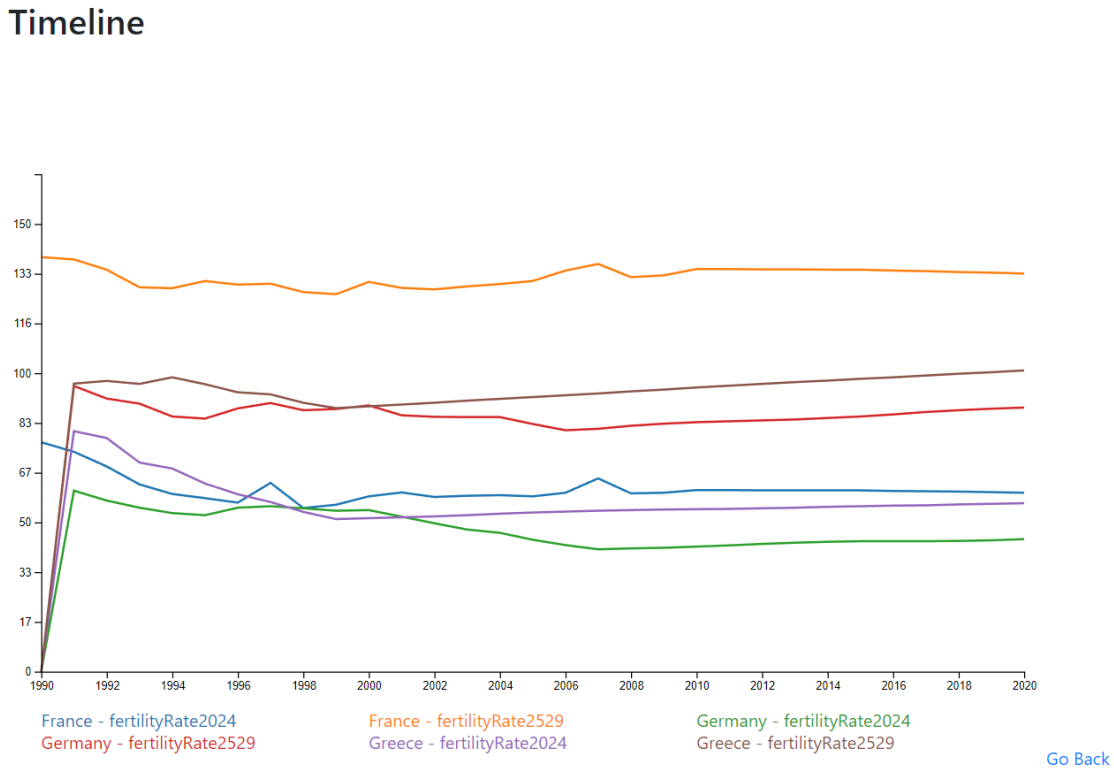
Picture 2.3 Class diagram

As we said this is a Spring boot application following the MVC pattern. So, for each table of the database we have a class to represent it in the main memory. If the primary key of a table contains more than one field then in Spring we represent the key with and extra

class ending with "…Id". The logic of the application is that the user selects countries and fields that he wants to see from a checklist and then the controller handles this request. In order to make the graph we pass to the html page a json string that contains everything that d3 library needs to make the graph. The logic of the creation of the json string that contains everything the user asked for in a proper form is created in the service layer of the application and depends on the type of graph we are about to present.

3 RESULTS STUDY

In the following screenshot , we can observe the fertility rates for the age groups of 20 to 24 and 25 to 29 in three countries: Greece, Germany and France



Based on this timeline , we can see that for the Age Group 20 to 24 :

- Over the years, the fertility rate in greece has exhibited a consistent and significant descrease
- Germany's fertility rate fluctuated between 40 and 65 during the same period
- France experienced a decline in fertility rate from 1990 to 1994 but then remained relatively stable between 58 and 65

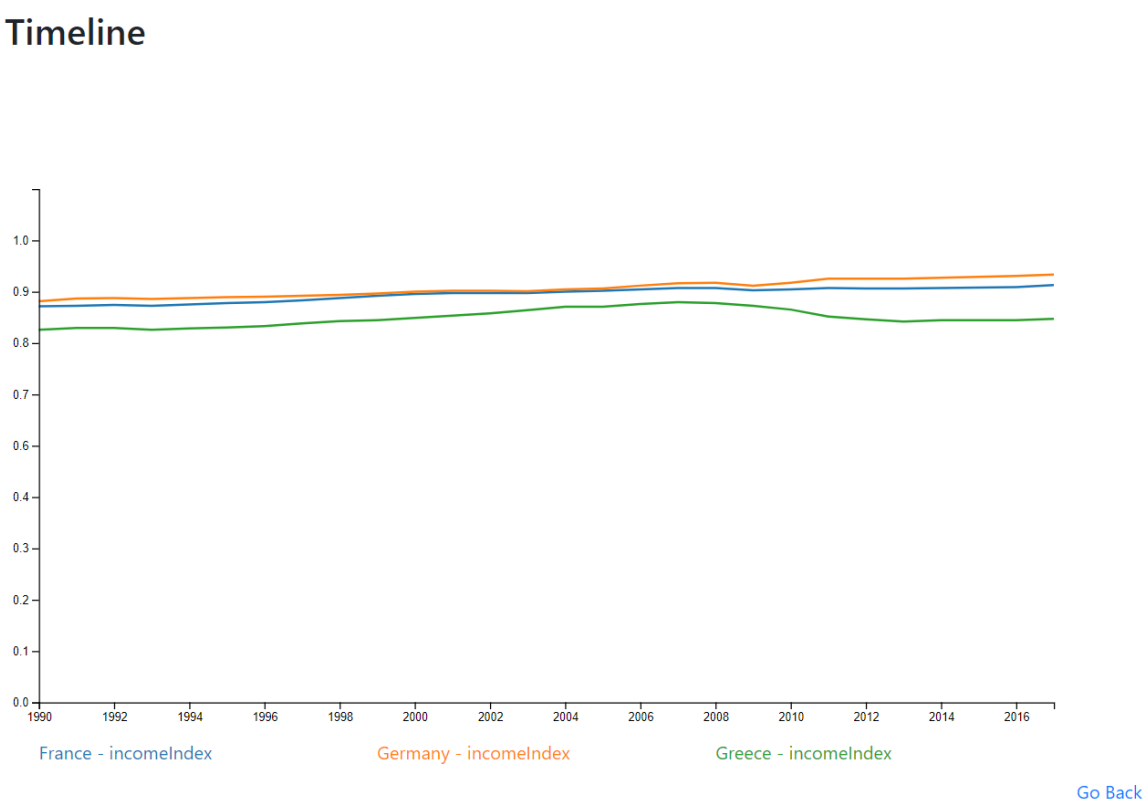
As for the Age Group 25 to 29 :

- Greece fertility rate for this group has a consistent inscrease from 1996 to 2020

- Germany's fertility rate decreased from 1990 to 1996 but remained relatively stable afterward, ranging between 80 to 100
- France fertility rate showed a slight decline in the early 1990s but then remained relatively stable between 128 and 137

Overall, it appears that France consistently had the highest fertility rates among the three countries for both age groups. Greece experienced an increase in fertility rates over the last few years, while Germany's rates remained relatively low.

In the following screenshot , we can see the income index for three countries: Greece, Germany and France



In Greece, the income index for Greece started at 0.808 in 1990 and gradually increased to 0.866 in 2008. However, after 2008, the income index experienced a slight decline, reaching 0.832 in 2017. Overall, Greece's income index displayed fluctuations and a slight downward trend over the years, indicating some economic challenges during the period.

As fot Germany ,its income index began at 0.87 in 1990 and consistently increased to 0.91 in 2008. After a slight decrease in 2009, the income index continued to rise, reaching 0.928 in 2017. The income index for Germany demonstrated a steady upward trend, suggesting sustained economic growth and stability throughout the analyzed period.

Finally, France's income index started at 0.859 in 1990 and remained relatively stable with minor variations. It reached its peak at 0.923 in 2015 and slightly decreased to

0.905 in 2017. The income index for France displayed a relatively stable pattern, indicating a consistent economic situation over the examined years.

However, based on the provided information, it is difficult to establish a direct correlation between economics and demographics in these countries.