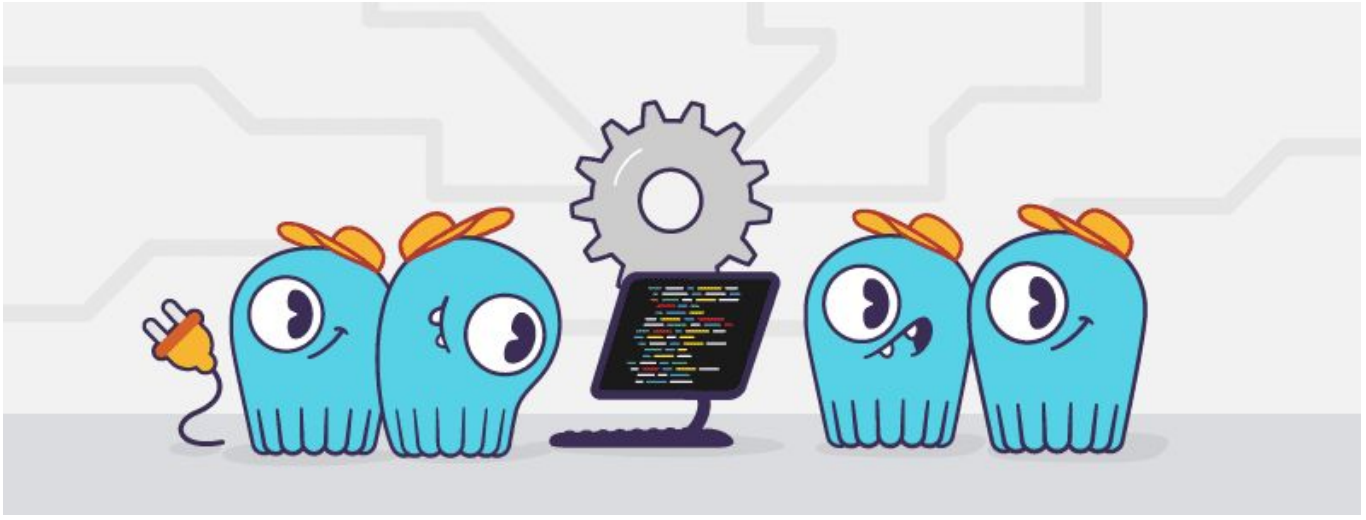
Groupe 1

Léa YOUSSEF
Arnaud MAGARIAN
Guilhem MAILLEBUAU
Marie-Alix HONORÉ
Yasmina BENDAOUD

Livrable 4 - Développement d'application Cloud

Application cloud



07/11/2020

Table des matières

I. Dénormalisation "_id" : {'year', 'country'}

```
{
        "_id":  { "year" : 2010 , "Country_name" : "France"}
        "country_stats":
        {
                "Country_area" : 652230,
                "Crude_birth_rate": 45.96,
                                "Crude_death_rate": 18.73,
                                "Net_migration": -9.88,
                                "Rate_natural_increase": 2.723,
                                "Growth_rate": 1.735,
                                "Fertility_rate_15-19": 124.2,
                "Fertility_rate_20-24": 291.9,
                "Fertility_rate_25-29": 328.1,
                "Fertility_rate_30-34": 277,
                "Fertility_rate_35-39": 218.4,
                "Fertility_rate_40-44": 129.2,
                "Fertility_rate_45-49": 71.2,
                "total_fertility_rate": 7.2,
                "gross_reproduction_rate": 3.5122,
                "sex_ratio_at_birth": 1.05,
                "Midyear_population": 22507460,
                "Infant_mortality": 22507460,
                "Life_expectancy": 45.81,
                "Mortality_rate_under5": 216.56,
                "Mortality_rate_1to4": 83.94,
                "Flag": [ {
                                        "total_flag": [A,*],
                        "Starting_age": 9,
                        "Ending_age": 15,
                        "Age_group_indicator": [+,-],
                        "Midyear_population_male": 1104463,
                        "Midyear_population_female": 1055773,
                        "Midyear_population": 2160236,
                }]
                       "Male":{
                        "Infant_mortality_male": 153.37,
                        "Life_expectancy_male": 44.85,
                        "Mortality_rate_under5_male": 222.96,
                        "Mortality_rate_1to4_male": 82.20,
                        "Max_age": 100,
                        "Population_age_0": 64496,
                        ….
                        "Population_age_100": 0,
                }
                          "Female"{
                        "Infant_mortality_female": 135.74,
                        "Life_expectancy_female": 46.83,
                        "Mortality_rate_under5_female": 209.84,
                        "Mortality_rate_1to4_male": 85.74,
                        "Max_age": 100,
                        "Population_age_0": 25166,
                        ….
                        "Population_age_100": 8484,
                }
        }
}
```

II. Importation des données

Afin de dénormaliser le dataset en respectant le schéma produit dans le 2ème livrable, nous avons développé un script python permettant, à partir des fichiers au format csv, de créer un unique fichier json. L'objectif de ce script était de rassembler toutes les informations dans une structure logique dans un dataframe.

Lors de l'importation de nos différents fichiers csv, nous avons procédé à un data cleaning, dans le but de conserver uniquement les informations intéressantes pour nos requêtes. Nous avons donc supprimé la colonne "country_code".

Nous avons fait le choix de négliger la table midyear_population_age_country car c'est une permutation de la table midyear_population_age_sex et d'une autre table que nous n'avons pas.

Nous avons rassemblé toutes nos données importées dans des dataframe dans un unique dataframe. Nous avons mergé les bases de données avec des outer-join sur les attributs country_name et year.

Ensuite, nous avons créé plusieurs listes:

- "flags": qui contient un tableau de flag pour chaque pays/année/sexe
- "sexe": qui contient deux tableaux avec les attributs correspondants à "Male" et "Female" pour respecter notre schéma de dénormalisation.

→ regrouper les deux informations "Male" et "Female" dans un même tableau nous permet d'obtenir un Dataframe ou chaque ligne correspond à [1 pays+1 année].

En parallèle, nous avons réalisé des modifications comme le passage de variables en variables de type str, le changement des virgules en points, etc.

Nous avons finalement créé des fonctions qui parcourent ce dataframe et ses tableaux pour générer un fichier json de 23080 documents.

Ensuite, dans Studio 3T, nous avons créé la base de données countrybase et des collections qui nous permettent d'appliquer les requêtes.

voir dénormalisation dans dossier.

III. Mesure de performances

Requêtes :

Nous avons explicité nos 8 questions sous forme de requêtes sur MongoDB.

- Requête 1 :

```
db.year_key.find({"_id.year": "2020", "country_stats.life_expectancy" : {$ne :
"null"} }).sort({"country_stats.life_expectancy": -1}).limit(1)
```

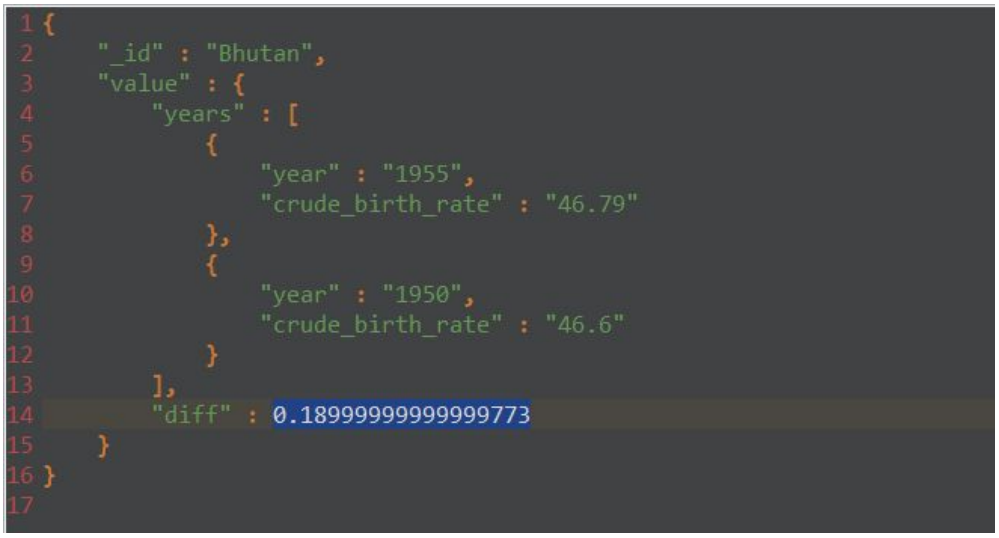
- Requête 2 :

```
db.Year_Key.aggregate([{$match : {"_id.year" : "2020"}},{$project :
{"_id.country_name": 1, "pop_km2" : { "$divide": [{$toDecimal:
"$country_stats.midyear_population"},
{$toDecimal:"$country_stats.country_area"}]}}}]);
```

Les requêtes suivantes utilisent toutes map reduce.

- Requête 3 :

```
var map2 = function(){
    if(this._id.year == "1955" || this._id.year == "1950")
        if( this.country_stats.crude_birth_rate != "null")
            emit(this._id.country_name, {"years":[{year : this._id.year,
crude_birth_rate : this.country_stats.crude_birth_rate}], diff:null });}
var reduce2 = function(key, values){
   y = {"years":[]};
   val55 = 0;
   val50 = 0;
   for(i =0 ; i<values.length;i++)
       for(j=0; j<values[i].years.length ; j++){
           if(values[i].years[j].year == "1950") val50 =
values[i].years[j].crude_birth_rate;
           if(values[i].years[j].year == "1955") val55 =
values[i].years[j].crude_birth_rate;
           y.years.push(values[i].years[j]);  }
      if(val55 > 0 && val50 > 0)
      y.diff = val55 - val50;
       else
              y.diff = null;
       return y; };
var queryParam2 ={ query: {},out: "map_reduce_baby_boom"};
db.year_key.mapReduce(
             map2,
             reduce2,
             queryParam2)
db.map_reduce_baby_boom.find({"value.diff" : {"$gte" : 0.1}});
```



```
{
    "_id" : "Bhutan",
    "value" : {
        "years" : [
            {
                "year" : "1955",
                "crude_birth_rate" : "46.79"
            },
            {
                "year" : "1950",
                "crude_birth_rate" : "46.6"
            }
        ],
        "diff" : 0.18999999999999773
    }
}
```

- Requête 4 :

```
var map4 = function(){
    if(this._id.country_name == "Iraq" || this._id.country_name == "United States"
)
        if(parseInt(this._id.year) > 1999 & parseInt(this._id.year) < 2016)
            if(this.country_stats.crude_birth_rate != "null" &&
this.country_stats.crude_death_rate != "null")
                emit(this._id.country_name, {"years":[{year : this._id.year,
crude_birth_rate : this.country_stats.crude_birth_rate, crude_death_rate :
this.country_stats.crude_death_rate, diff: null}] }); };
var reduce4 = function(key, values){
   y = {"years":[]};
   for(i =0 ; i<values.length;i++)
       for(j=0; j<values[i].years.length ; j++)
       {
           values[i].years[j].diff = values[i].years[j].crude_birth_rate -
values[i].years[j].crude_death_rate;
           y.years.push(values[i].years[j])    }
    return y ;   };
db.year_key.explain("executionStats").mapReduce(
        map4,
        reduce4, {
            query: {},
            out: "map_reduce_impact_irak"  }
    )
db.map_reduce_impact_irak.find();
```

- Requête 5

```
var map5 = function(){
    if((this._id.year =="2020" || this._id.year == "2010") &&
this.country_stats.crude_birth_rate!=null )
        emit(this._id.country_name, {"years":[{year : this._id.year,
crude_birth_rate : this.country_stats.crude_birth_rate}], diff:null}); };
var reduce5 = function(keyCountry, values) {
    y = {"years":[]};
   val55 = 0;
   val50 = 0;
   for(i =0 ; i<values.length;i++)
       for(j=0; j<values[i].years.length ; j++){
           if(values[i].years[j].year == "2020") val55 =
values[i].years[j].crude_birth_rate;
           if(values[i].years[j].year == "2010") val50 =
values[i].years[j].crude_birth_rate;
           y.years.push(values[i].years[j]);
       }
    if(val55 > 0 && val50 > 0)
        y.diff = val55 - val50;
    else
        y.diff = null;
   return y; }

db.Year_Key.mapReduce(
       map5,
       reduce5,{
             query: {},
             out: "map_reduce_max_increase" })
db.map_reduce_max_increase.find().sort({"value.diff":-1}).limit(1)
```

- Requête 6 :

```
var map6 = function(){
    if(parseInt(this._id.year) > 2010 && parseInt(this._id.year) < 2021)
        if (this.country_stats.rate_natural_increase != "null")
            emit(this._id.country_name, {"years":[{year : this._id.year,
natural_rate : parseFloat(this.country_stats.rate_natural_increase), population :
this.country_stats.midyear_population}], avg_evolution:null, pop_decroiss :
null});
}
var reduce6 = function(key, values)
{
   y = {"years":[]};
   val19 = 0;
   val20 = 0;
   sum_natural_rate = 0;
   count = 0;
   for(i =0 ; i<values.length;i++)
       for(j=0; j<values[i].years.length ; j++){
           if(values[i].years[j].year == "2019") val19 =
values[i].years[j].population;
           if(values[i].years[j].year == "2020") val20 =
values[i].years[j].population;
           if(values[i].years[j].natural_rate != null)
               sum_natural_rate += values[i].years[j].natural_rate;
               count ++;
           y.years.push(values[i].years[j]);
       }
```

```
    if(sum_natural_rate != 0)
        y.avg_evolution = sum_natural_rate/count;
    else
        y.avg_evolution = null;
    if(val19 > 0 && val20 > 0)
        if( val20 - val19 < 0)
            y.pop_decroiss = "Décroissante";
        else
            y.pop_decroiss = "Croissante";
    else
        y.pop_decroiss = null;
   return y;
};
var queryParam6 ={ query: {},out: "map_reduce_baby_boom"};
db.year_key.mapReduce(
             map6,
             reduce6,
             queryParam6
       )
db.map_reduce_baby_boom.find({"value.avg_evolution" : {$ne :
null}}).sort({"value.avg_evolution":1}).limit(10);
```

- Requête 7

```
var map7 = function()
{
    if(this.country_stats.mortality_rate_under5 != "null" )
        emit(this._id.year, {"countries" :[{country : this._id.country_name, mru
:this.country_stats.Male.infant_mortality_male }] ,count:null});
};
var reduce7 = function(key, values)
{
    y = {"countries":[]}
    var country_count = 0
    for(i =0; i< values.length; i++){
        if(values[i].countries !=null) {
            for(j=0; j<values[i].countries.length ; j++){
                if(parseFloat(values[i].countries[j].mru) >100.0) {
                    country_count = country_count +1
                    y.countries.push(values[i].countries[j]) } } }
    }
    y.count = country_count
    return y;  };
db.year_key.mapReduce(
        map7,
        reduce7,
        {
            query: {},
            out: "map_reduce_max_increase"
        }
    )
db.map_reduce_max_increase.find().sort({"_id.year":-1});
```

- Requête 8 :

```
var map8 = function()
{
   if(this._id.country_name == "France")
      emit(this._id.year, {"populations":[{male_pop : this.country_stats.Male, female_pop :
this.country_stats.Female }],max: null}); };
var reduce8 = function(key, values){
   y = {"populations":[]}
   index = [];
   for(i =0; i< values.length; i++)
   {
      if(values[i].populations !=null)
       {
         for(j=0; j<values[i].populations.length ; j++)
         {
            if (values[i].populations[j] !=null)
          {     max = 0.0;
                if (values[i].populations[j].male_pop != null & values[i].populations[j].female_pop != null)
               {
                y.populations.push(values[i].populations[j]);
                index.push(parseFloat(values[i].populations[j].male_pop.population_age_0) +
parseFloat(values[i].populations[j].female_pop.population_age_0))
                index.push(parseFloat(values[i].populations[j].male_pop.population_age_1) +
parseFloat(values[i].populations[j].female_pop.population_age_1))
                index.push(parseFloat(values[i].populations[j].male_pop.population_age_2) +
parseFloat(values[i].populations[j].female_pop.population_age_2))
                index.push(parseFloat(values[i].populations[j].male_pop.population_age_3) +
parseFloat(values[i].populations[j].female_pop.population_age_3))
                index.push(parseFloat(values[i].populations[j].male_pop.population_age_4) +
parseFloat(values[i].populations[j].female_pop.population_age_4))
                index.push(parseFloat(values[i].populations[j].male_pop.population_age_5) +
parseFloat(values[i].populations[j].female_pop.population_age_5))
                index.push(parseFloat(values[i].populations[j].male_pop.population_age_6) +
parseFloat(values[i].populations[j].female_pop.population_age_6))
                index.push(parseFloat(values[i].populations[j].male_pop.population_age_7) +
parseFloat(values[i].populations[j].female_pop.population_age_7))
                index.push(parseFloat(values[i].populations[j].male_pop.population_age_8) +
parseFloat(values[i].populations[j].female_pop.population_age_8))
                index.push(parseFloat(values[i].populations[j].male_pop.population_age_9) +
parseFloat(values[i].populations[j].female_pop.population_age_9))
                index.push(parseFloat(values[i].populations[j].male_pop.population_age_10) +
parseFloat(values[i].populations[j].female_pop.population_age_10))
                index.push(parseFloat(values[i].populations[j].male_pop.population_age_11) +
parseFloat(values[i].populations[j].female_pop.population_age_11))
                index.push(parseFloat(values[i].populations[j].male_pop.population_age_12) +
parseFloat(values[i].populations[j].female_pop.population_age_12))
                index.push(parseFloat(values[i].populations[j].male_pop.population_age_13) +
parseFloat(values[i].populations[j].female_pop.population_age_13))
                index.push(parseFloat(values[i].populations[j].male_pop.population_age_14) +
parseFloat(values[i].populations[j].female_pop.population_age_14))
                index.push(parseFloat(values[i].populations[j].male_pop.population_age_15) +
parseFloat(values[i].populations[j].female_pop.population_age_15))
                index.push(parseFloat(values[i].populations[j].male_pop.population_age_16) +
parseFloat(values[i].populations[j].female_pop.population_age_16))
                index.push(parseFloat(values[i].populations[j].male_pop.population_age_17) +
parseFloat(values[i].populations[j].female_pop.population_age_17))
```

```
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_18) +
parseFloat(values[i].populations[j].female_pop.population_age_18))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_19) +
parseFloat(values[i].populations[j].female_pop.population_age_19))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_20) +
parseFloat(values[i].populations[j].female_pop.population_age_20))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_21) +
parseFloat(values[i].populations[j].female_pop.population_age_21))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_22) +
parseFloat(values[i].populations[j].female_pop.population_age_22))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_23) +
parseFloat(values[i].populations[j].female_pop.population_age_23))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_24) +
parseFloat(values[i].populations[j].female_pop.population_age_24))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_25) +
parseFloat(values[i].populations[j].female_pop.population_age_25))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_26) +
parseFloat(values[i].populations[j].female_pop.population_age_26))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_27) +
parseFloat(values[i].populations[j].female_pop.population_age_27))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_28) +
parseFloat(values[i].populations[j].female_pop.population_age_28))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_29) +
parseFloat(values[i].populations[j].female_pop.population_age_29))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_30) +
parseFloat(values[i].populations[j].female_pop.population_age_30))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_31) +
parseFloat(values[i].populations[j].female_pop.population_age_31))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_32) +
parseFloat(values[i].populations[j].female_pop.population_age_32))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_33) +
parseFloat(values[i].populations[j].female_pop.population_age_33))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_34) +
parseFloat(values[i].populations[j].female_pop.population_age_34))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_35) +
parseFloat(values[i].populations[j].female_pop.population_age_35))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_36) +
parseFloat(values[i].populations[j].female_pop.population_age_36))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_37) +
parseFloat(values[i].populations[j].female_pop.population_age_37))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_38) +
parseFloat(values[i].populations[j].female_pop.population_age_38))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_39) +
parseFloat(values[i].populations[j].female_pop.population_age_39))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_40) +
parseFloat(values[i].populations[j].female_pop.population_age_40))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_41) +
parseFloat(values[i].populations[j].female_pop.population_age_41))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_42) +
parseFloat(values[i].populations[j].female_pop.population_age_42))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_43) +
parseFloat(values[i].populations[j].female_pop.population_age_43))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_44) +
parseFloat(values[i].populations[j].female_pop.population_age_44))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_45) +
parseFloat(values[i].populations[j].female_pop.population_age_45))
```

```
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_46) +
parseFloat(values[i].populations[j].female_pop.population_age_46))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_47) +
parseFloat(values[i].populations[j].female_pop.population_age_47))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_48) +
parseFloat(values[i].populations[j].female_pop.population_age_48))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_49) +
parseFloat(values[i].populations[j].female_pop.population_age_49))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_50) +
parseFloat(values[i].populations[j].female_pop.population_age_50))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_51) +
parseFloat(values[i].populations[j].female_pop.population_age_51))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_52) +
parseFloat(values[i].populations[j].female_pop.population_age_52))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_53) +
parseFloat(values[i].populations[j].female_pop.population_age_53))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_54) +
parseFloat(values[i].populations[j].female_pop.population_age_54))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_55) +
parseFloat(values[i].populations[j].female_pop.population_age_55))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_56) +
parseFloat(values[i].populations[j].female_pop.population_age_56))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_57) +
parseFloat(values[i].populations[j].female_pop.population_age_57))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_58) +
parseFloat(values[i].populations[j].female_pop.population_age_58))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_59) +
parseFloat(values[i].populations[j].female_pop.population_age_59))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_60) +
parseFloat(values[i].populations[j].female_pop.population_age_60))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_61) +
parseFloat(values[i].populations[j].female_pop.population_age_61))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_62) +
parseFloat(values[i].populations[j].female_pop.population_age_62))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_63) +
parseFloat(values[i].populations[j].female_pop.population_age_63))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_64) +
parseFloat(values[i].populations[j].female_pop.population_age_64))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_65) +
parseFloat(values[i].populations[j].female_pop.population_age_65))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_66) +
parseFloat(values[i].populations[j].female_pop.population_age_66))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_67) +
parseFloat(values[i].populations[j].female_pop.population_age_67))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_68) +
parseFloat(values[i].populations[j].female_pop.population_age_68))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_69) +
parseFloat(values[i].populations[j].female_pop.population_age_69))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_70) +
parseFloat(values[i].populations[j].female_pop.population_age_70))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_71) +
parseFloat(values[i].populations[j].female_pop.population_age_71))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_72) +
parseFloat(values[i].populations[j].female_pop.population_age_72))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_73) +
parseFloat(values[i].populations[j].female_pop.population_age_73))
```

```
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_74) +
parseFloat(values[i].populations[j].female_pop.population_age_74))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_75) +
parseFloat(values[i].populations[j].female_pop.population_age_75))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_76) +
parseFloat(values[i].populations[j].female_pop.population_age_76))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_77) +
parseFloat(values[i].populations[j].female_pop.population_age_77))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_78) +
parseFloat(values[i].populations[j].female_pop.population_age_78))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_79) +
parseFloat(values[i].populations[j].female_pop.population_age_79))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_80) +
parseFloat(values[i].populations[j].female_pop.population_age_80))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_81) +
parseFloat(values[i].populations[j].female_pop.population_age_81))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_82) +
parseFloat(values[i].populations[j].female_pop.population_age_82))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_83) +
parseFloat(values[i].populations[j].female_pop.population_age_83))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_84) +
parseFloat(values[i].populations[j].female_pop.population_age_84))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_85) +
parseFloat(values[i].populations[j].female_pop.population_age_85))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_86) +
parseFloat(values[i].populations[j].female_pop.population_age_86))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_87) +
parseFloat(values[i].populations[j].female_pop.population_age_87))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_88) +
parseFloat(values[i].populations[j].female_pop.population_age_88))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_89) +
parseFloat(values[i].populations[j].female_pop.population_age_89))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_90) +
parseFloat(values[i].populations[j].female_pop.population_age_90))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_91) +
parseFloat(values[i].populations[j].female_pop.population_age_91))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_92) +
parseFloat(values[i].populations[j].female_pop.population_age_92))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_93) +
parseFloat(values[i].populations[j].female_pop.population_age_93))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_94) +
parseFloat(values[i].populations[j].female_pop.population_age_94))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_95) +
parseFloat(values[i].populations[j].female_pop.population_age_95))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_96) +
parseFloat(values[i].populations[j].female_pop.population_age_96))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_97) +
parseFloat(values[i].populations[j].female_pop.population_age_97))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_98) +
parseFloat(values[i].populations[j].female_pop.population_age_98))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_99) +
parseFloat(values[i].populations[j].female_pop.population_age_99))
            index.push(parseFloat(values[i].populations[j].male_pop.population_age_100) +
parseFloat(values[i].populations[j].female_pop.population_age_100))
} } } } }
    var i = index.indexOf(Math.max(...index));
   y.max = i;
```

```
   return y;  };
db.year_key.mapReduce(
      map8,
      reduce8,
      {
         query: {},
         out: "map_reduce_fr_most_represented_age_group"
      }
   )
db.map_reduce_fr_most_represented_age_group.find({"value.max" : {$ne :-1.0}});
```

Mesure des temps d'exécution :

Afin de calculer le temps d'exécution, nous avons utilisé un tableau Excel pour faire les calculs (voir ci-dessous).
Pour chaque requête, pour les 6 shards, nous avons noté le temps d'exécution pour chaque réplicaset.
Nous en avons ensuite fait la somme, et nous avons répété l'opération 10 fois de suite.
Enfin, nous avons calculé la moyenne sur ces 10 valeurs en retirant la valeur minimale et maximale (tableau de droite).
Nous avons répété l'opération en retirant un shard c'est-à-dire que nous avons calculé le temps avec une répartition sur 5 shards, puis 4 shards, puis 3 shards etc... jusqu'à tester sur 1 shard.

Temps d'éxecution

Répartition sur 6 shards

Somme des temps d'exécution en Millis des shards pars requête							
Requête -shards	1	2	3	4	5	6	Somme
R1	1232,0						1232,0
R2	0,0						0,0
R3	46,0	49,0	49,0	53,0	62,0	71,0	330,0
R4	30,0	41,0	48,0	57,0	62,0	76,0	314,0
R5	37,0	37,0	53,0	54,0	62,0	90,0	333,0
R6	36,0	46,0	50,0	57,0	68,0	143,0	400,0
R7	45,0	65,0	69,0	125,0	139,0	169,0	612,0
R8	43,0	48,0	63,0	70,0	86,0	95,0	405,0

Calcul de la moyenne sur 10 itérations										
1	2	3	4	5	6	7	8	9	10	Moyenne[total-min-max]
1232,00	26,00	3,00	3,00	2,00	3,00	2,00	2,00	2,00	3,00	5,50
0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
327,00	316,00	327,00	312,00	319,00	325,00	315,00	326,00	320,00	330,00	321,88
329,00	332,00	327,00	324,00	336,00	343,00	331,00	317,00	316,00	314,00	326,50
339	338	340	348	338	348	332	342	350	333	340,75
376	413	392	388	395	403	388	399	394	400	394,88
602	605	600	585	581	600	593	618	602	612	599,88
6651,00	392,00	395,00	405,00	399,00	409,00	429,00	402,00	394,00	405,00	404,75

Répartition sur 5 shards

Somme des temps d'exécution en Millis des shards pars requête							
Requête -shards	1	2	3	4	5	6	Somme
R1							0,0
R2							0,0
R3	46,0	47,0	66,0	64,0	68,0		291,0
R4	43,0	53,0	66,0	72,0	77,0		311,0
R5	41,0	47,0	64,0	65,0	77,0		294,0
R6	49,0	50,0	63,0	68,0	117,0		347,0
R7	46,0	79,0	138,0	157,0	169,0		589,0
R8	49,0	54,0	85,0	91,0	100,0		379,0

Calcul de la moyenne sur 10 itérations										
1	2	3	4	5	6	7	8	9	10	Moyenne[total-min-max]
4,00	6,00	4,00	3,00	4,00	4,00	4,00	5,00	5,00	5,00	4,38
1,00	1,00	0,00	0,00	0,00	1,00	1,00	1,00	1,00	1,00	0,75
310,00	307,00	304,00	300,00	311,00	312,00	299,00	302,00	301,00	291,00	304,25
258,00	309,00	309,00	309,00	302,00	302,00	302,00	315,00	318,00	310,00	307,25
307	315	321	320	330	333	307	301	299	294	312,50
351	362	377	350	349	349	347	335	358	347	351,63
553	554	564	547	549	568	566	564	567	589	560,63
396,00	383,00	385,00	376,00	369,00	390,00	370,00	386,00	396,00	379,00	383,13

Visualisation du tableau de coût pour 6 shards et 5 shards (le fichier excel contenant l'intégralité est dans le dossier du projet)

Observations et explications :

Moyenne(total-min)
5,50
0,00
321,88
326,50
340,75
394,88
599,88
404,75
2394,13

Moyenne(total-min)
4,38
0,75
304,25
307,25
312,50
351,63
560,63
383,13
2224,50

Moyenne(total-min)
4,38
0,75
279,75
272,25
294,00
331,00
535,13
358,13
2075,38

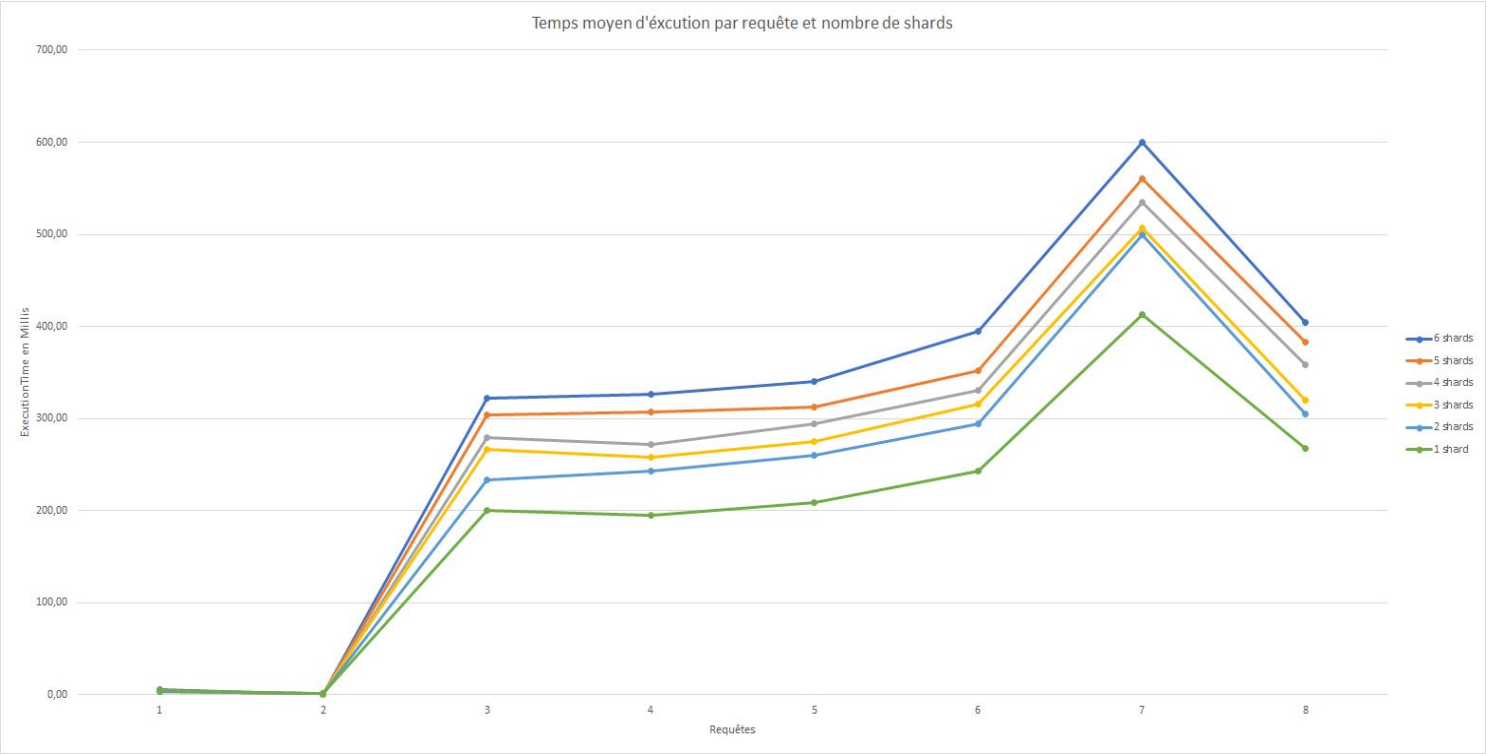
Moyenne(total-min)
4,13
1,00
266,13
258,00
275,13
316,13
507,13
319,75
1947,38

Moyenne(total-min)
4,13
1,00
233,00
242,88
260,25
293,88
499,38
304,75
1839,25

Moyenne(total-min)
4,38
1,00
200,38
195,25
208,38
243,25
412,63
267,25
1532,50

Pour chaque configuration, après avoir calculé le temps moyen au bout de 10 répétitions, nous avons additionné les différents temps moyen trouvés.
Nous pouvons constater qu'à mesure que le nombre de shards diminue, le temps d'exécution des requêtes diminue lui aussi fortement.

Ci-dessous : Une représentation graphique de la durée d'exécution des requêtes en fonction de celles-ci et du nombre de shards:



On voit que les courbes pour un même nombre de shards suivent la même tendance.
Le temps d'exécution est constant pour les deux premières requêtes qui sont simples et dont les données sont réparties sur un shard uniquement. Ce qui est la raison principale d'un temps d'exécution si bas.

Le temps d'exécution augmente beaucoup plus pour la requête 3 qui a besoin d'un coût plus élevé car les données sont répartis sur les 6 shards.
Globalement, l'allure continue d'augmenter avec les requêtes, on remarque une nette augmentation de coût pour la requête 7. La raison de ceci est que la requête va devoir appeler l'intégralité des shards pour ensuite réaliser un shuffle globale.

Comme précisé précédemment, pour une requête donnée, le temps d'exécution est plus élevé quand le nombre de shards augmente. Ce qui est compréhensible, car les données sont répartie sous différents shards et nécessite d'autant plus de coût réseau.

IV. Vues

Pour le développement de cette application, nous avons choisi Python et Django, pour la partie web et aussi pour la connexion à la base de données MongoDB en SSH.

Il y a 3 vues principales, une pour chaque type d'utilisateur et une vue pour la page d'accueil de notre application. Chaque utilisateur peut exécuter une liste de requêtes correspondant à son poste.

Le fichier urls.py contient une liste de chemins d'accès permettant de lier un URL à une fonction du fichier views.py.

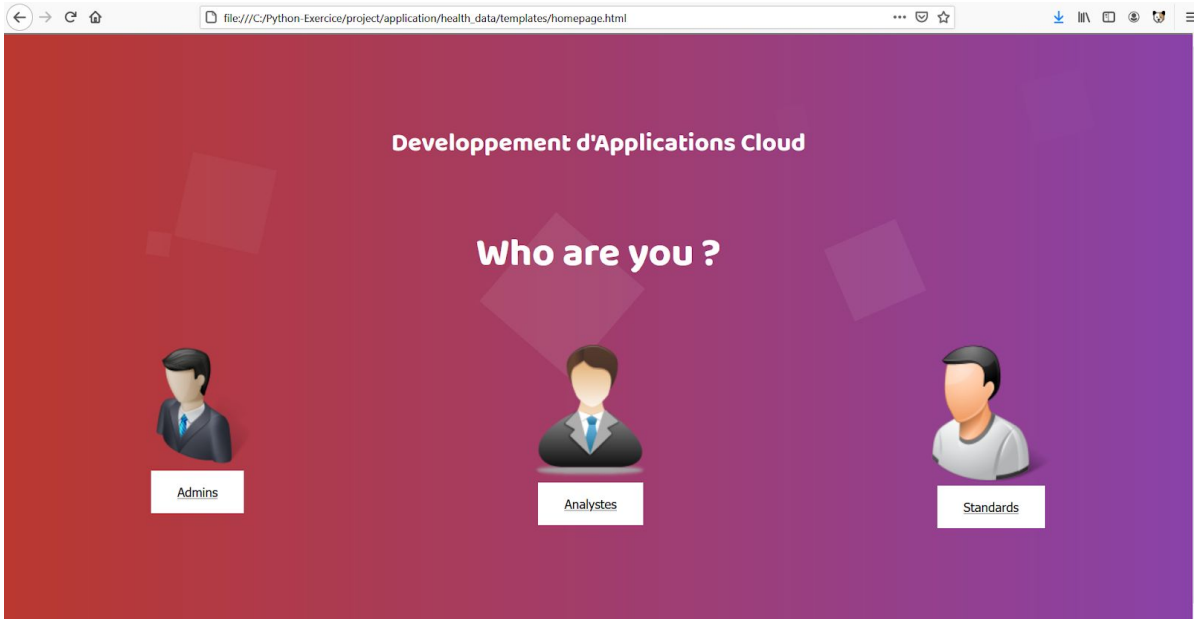
Dans le fichier views.py, on retrouve les diverses fonctions correspondant à nos différentes requêtes. Ces différentes fonctions retournent un HTML ainsi que les données que nous a retourné notre requête. Pour chaque fonction de requête on trouve un fichier HTML pour une vue adaptée.

On trouve aussi une fonction permettant de se connecter en SSH à notre base donnée ainsi que deux fonctions permettant de générer des graphs pour ensuite les afficher dans nos pages HTML.

0. Page d'accueil

Cette page permet d'accueillir le visiteur de l'application et propose le choix entre 3 boutons: Utilisateur standard, Utilisateur Décisionnaire et Administrateur.

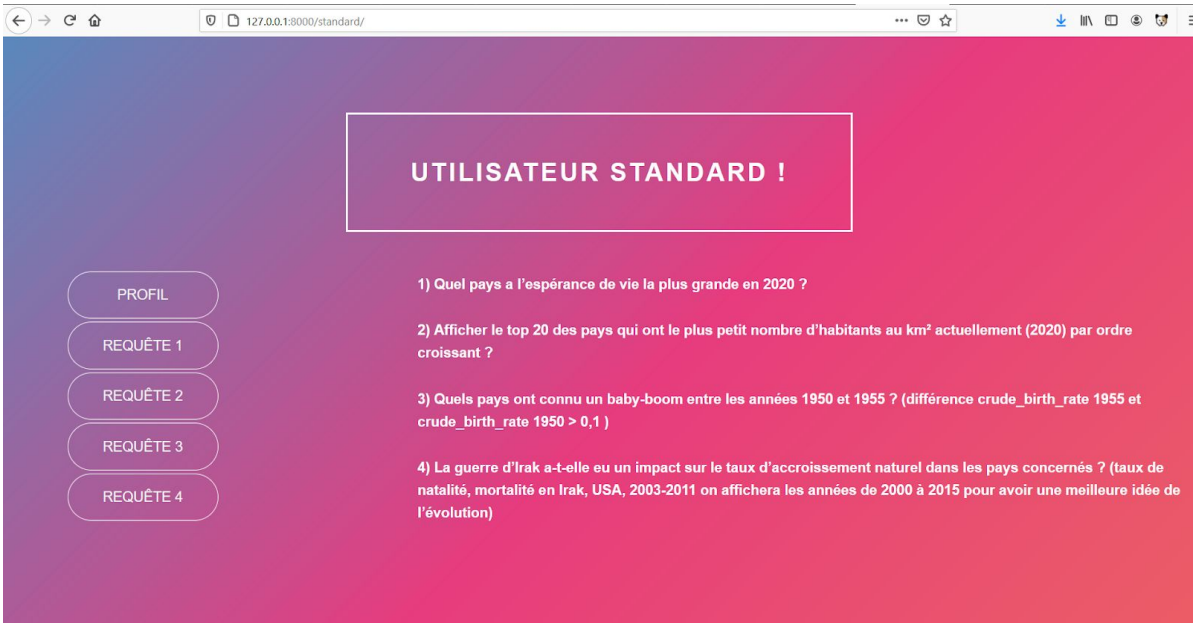
Ensuite, chaque vue est adaptée à ce que peut faire ou voir le visiteur et comporte les fonctionnalités associées, détaillées dans les parties suivantes



Page d'Accueil de l'application

1. Utilisateur Standard

Cet utilisateur peut accéder au 4 requêtes simples et voir le contenu des données à travers l'affichage.

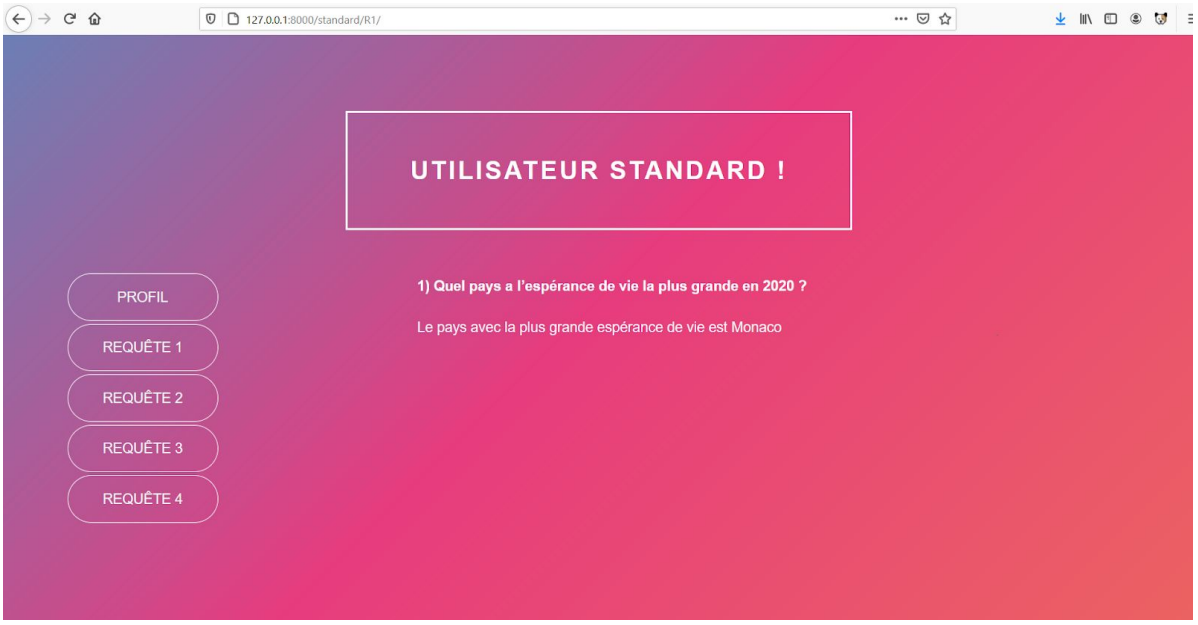


Page Utilisateur Standard

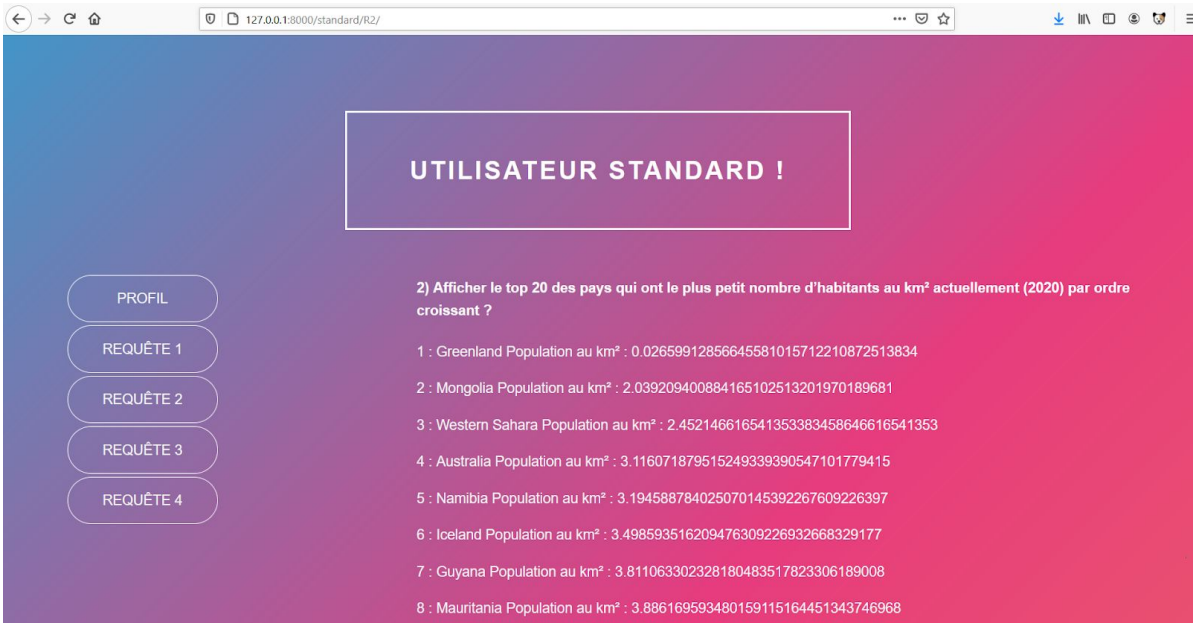
Grâce à des sélections sur le côté gauche, l'utilisateur standard peut avoir accès aux 4 requêtes simples et sur la droite, il peut voir les questions associées.

L'accès à chaque requête lui affiche la question associée à celle-ci ainsi qu'une phrase qui lui affiche le résultat, après s'être au préalable connecté à la base de données.

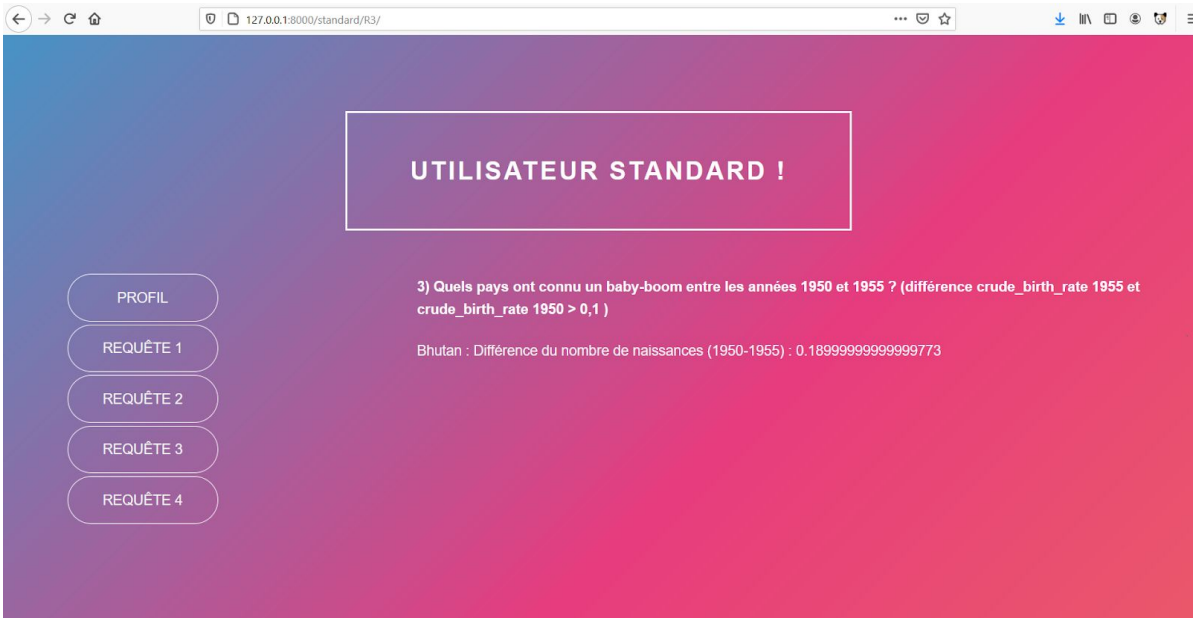
Il peut aussi mettre la souris sur "Profil", un sous menu s'affiche alors pour proposer de changer de statut.



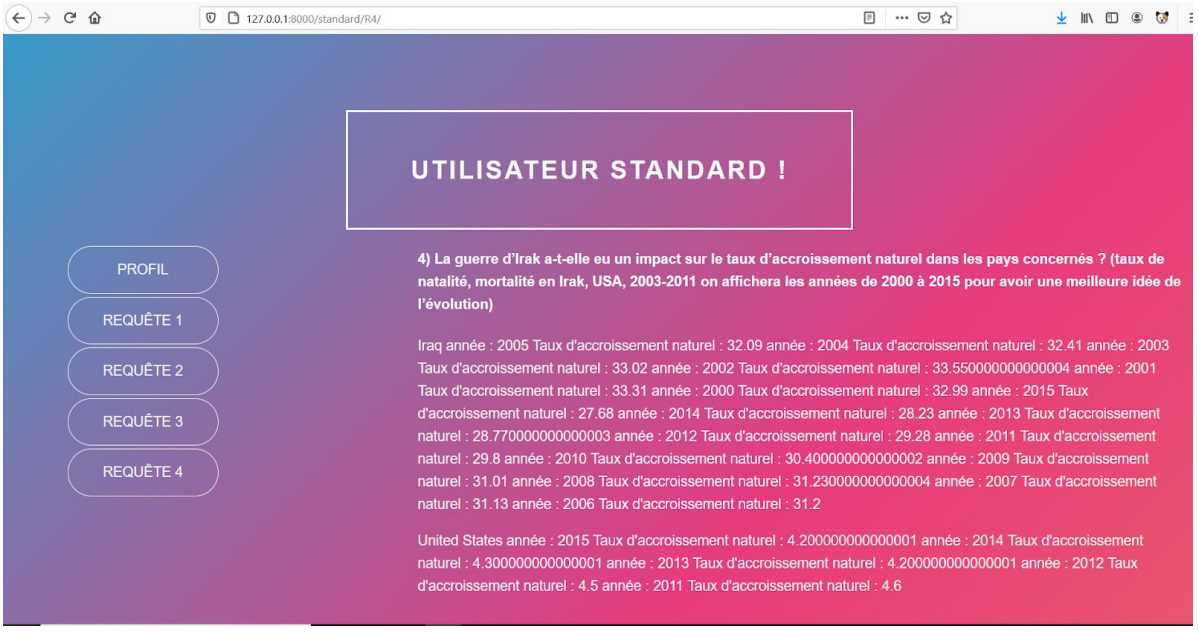
Bouton Requête 1



Bouton Requête 2



Bouton Requête 3



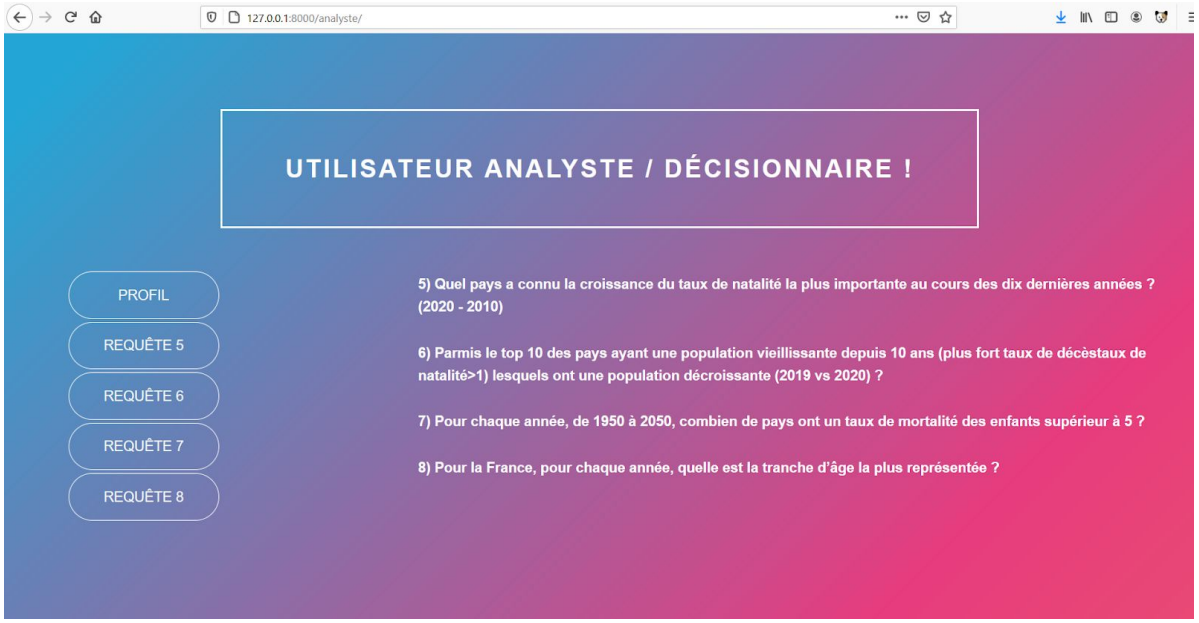
Bouton Requête 4

2. Analyste / Décisionnaire

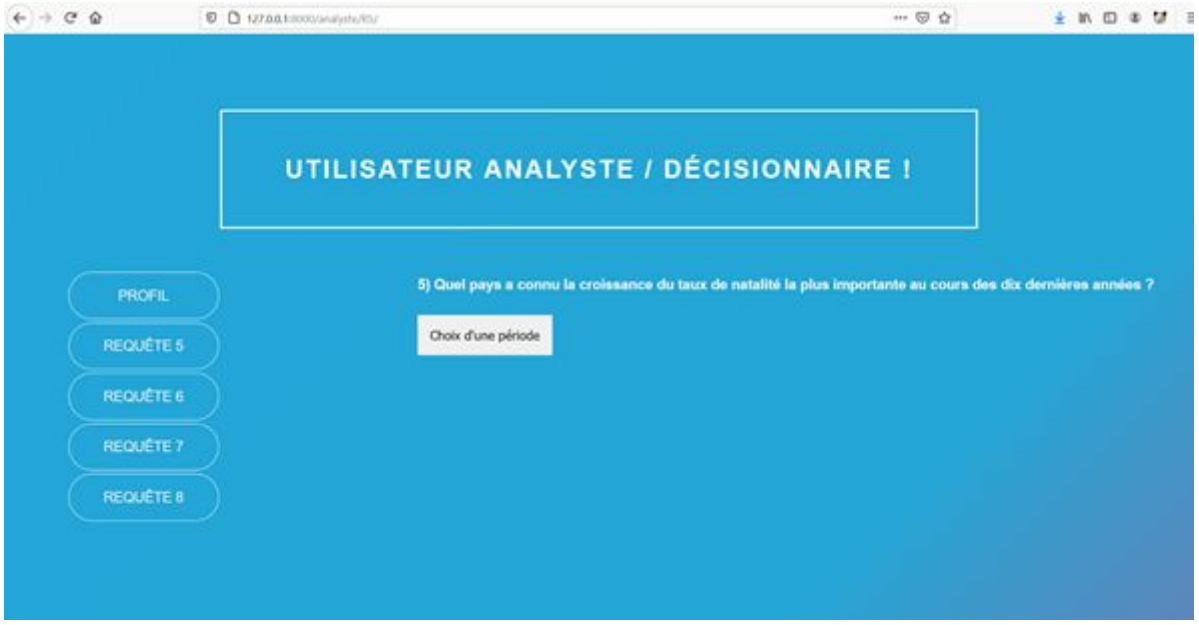
Le Data Analyst ou Business User a accès aux requêtes complexes proposées.

L'interface devra permettre de paramétrer les requêtes avec des valeurs par menu déroulant.

Si l'utilisateur choisit l'option "Analayste Décisionnaire", il arrive sur cette page :

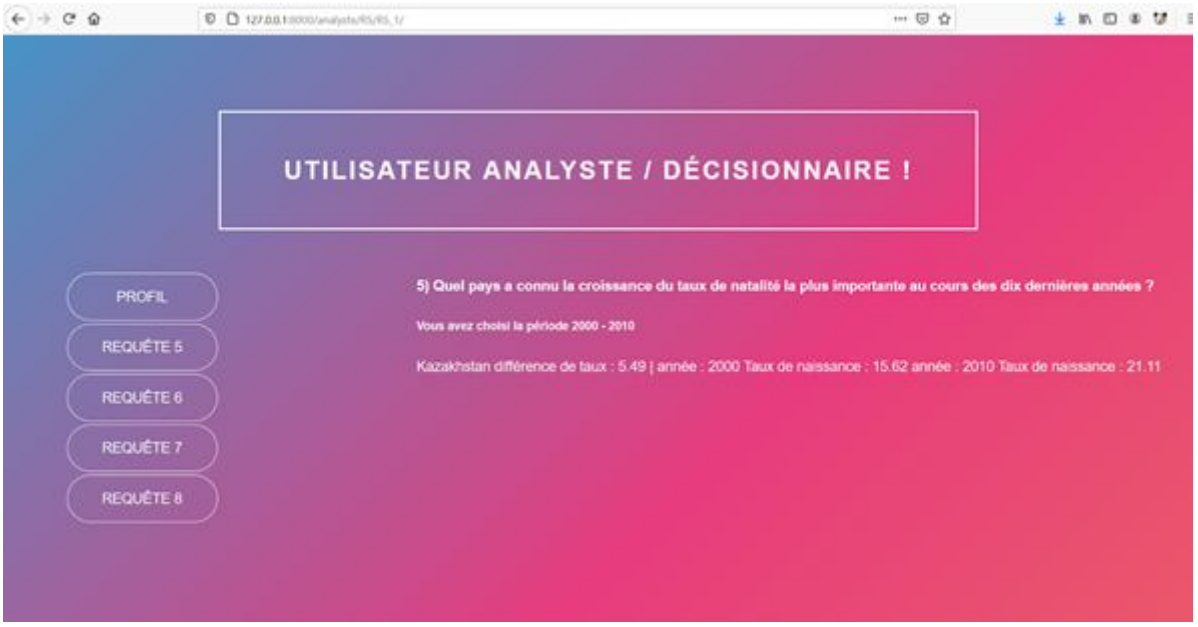


Il peut alors consulter les 4 requêtes difficiles depuis le menu.

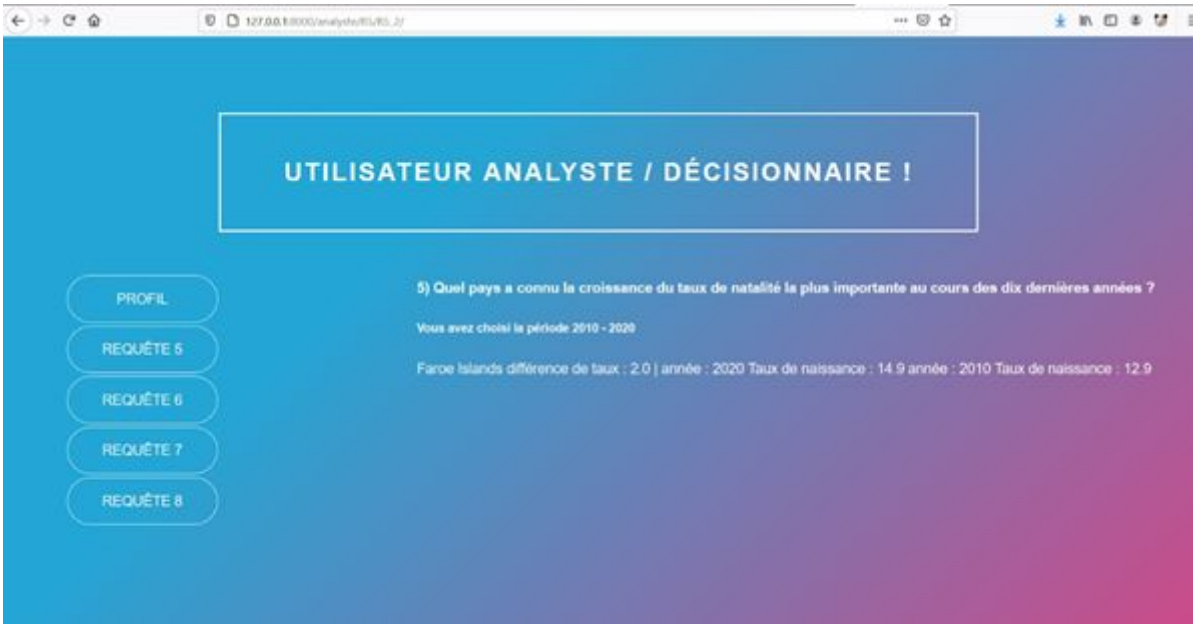


Bouton Requête 5

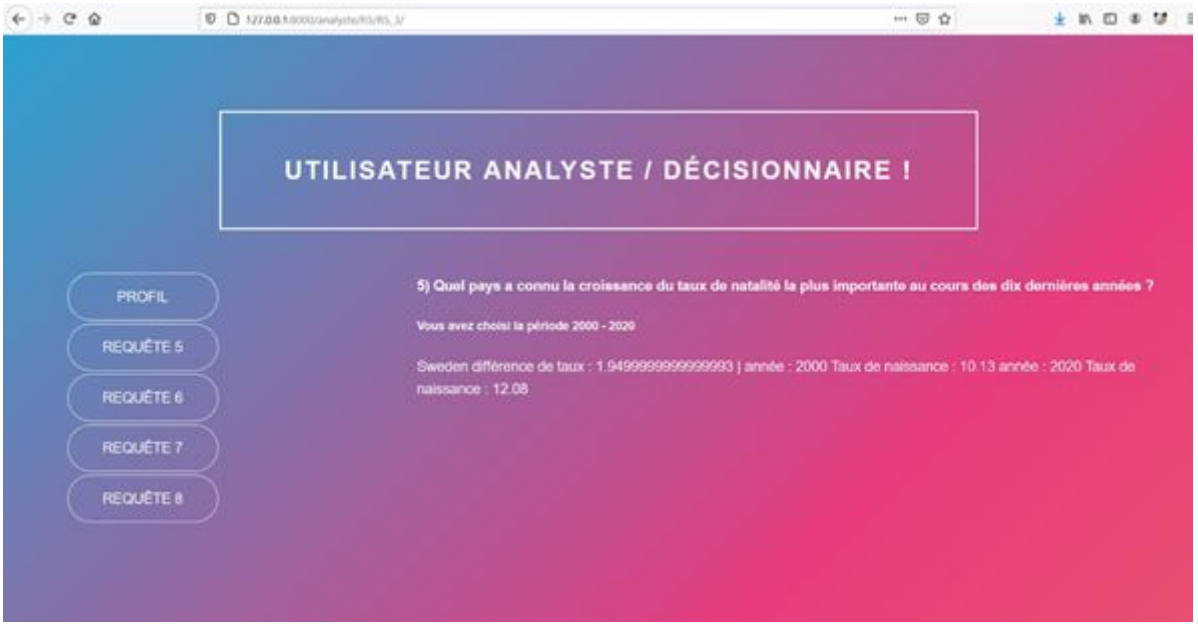
Pour cette requête, l'analyste peut paramétrer la requête en choisissant la période à appliquer.



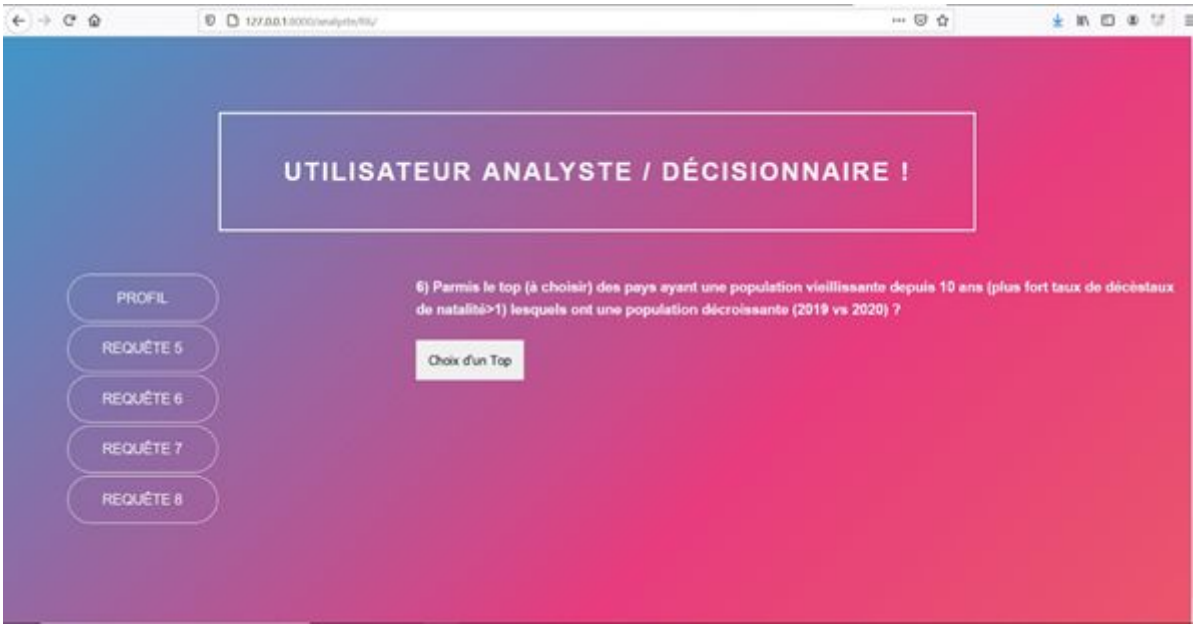
Résultat de 5) pour la sélection “2000-2010”



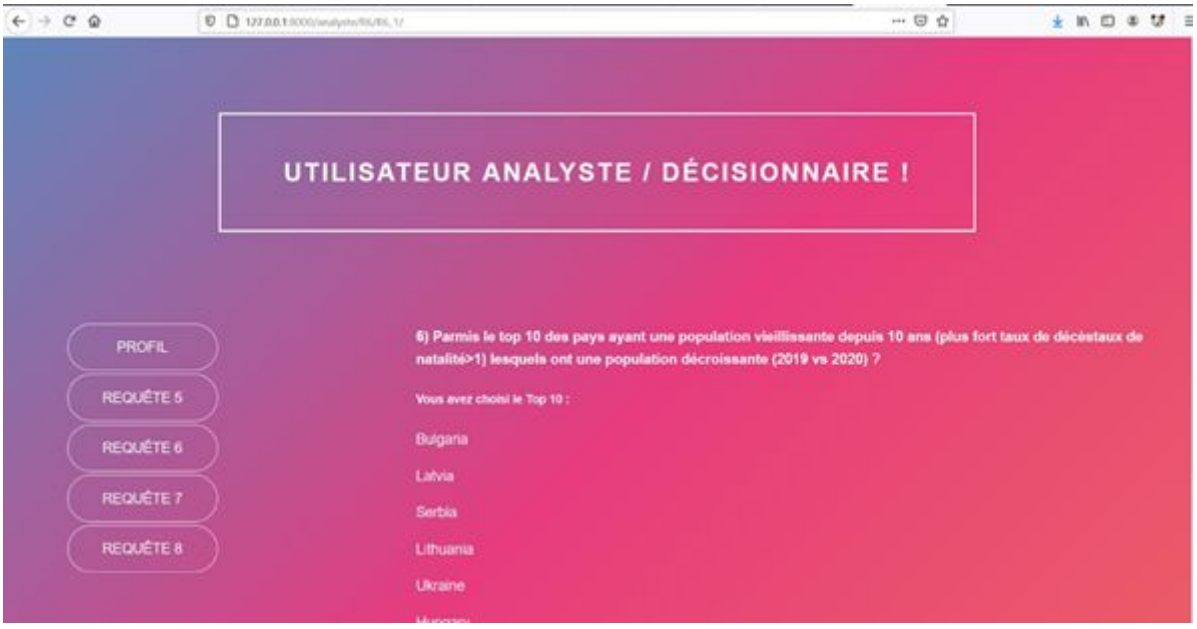
Résultat de 5) pour la sélection “2010-2020”



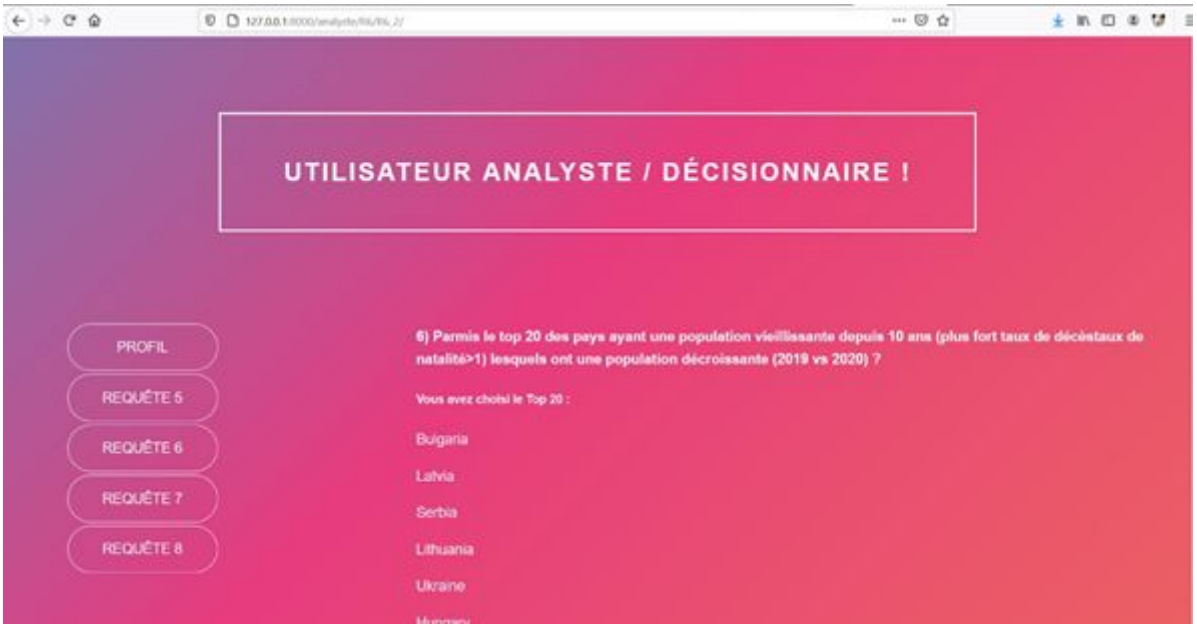
Résultat de 5) pour la sélection “2000-2020”



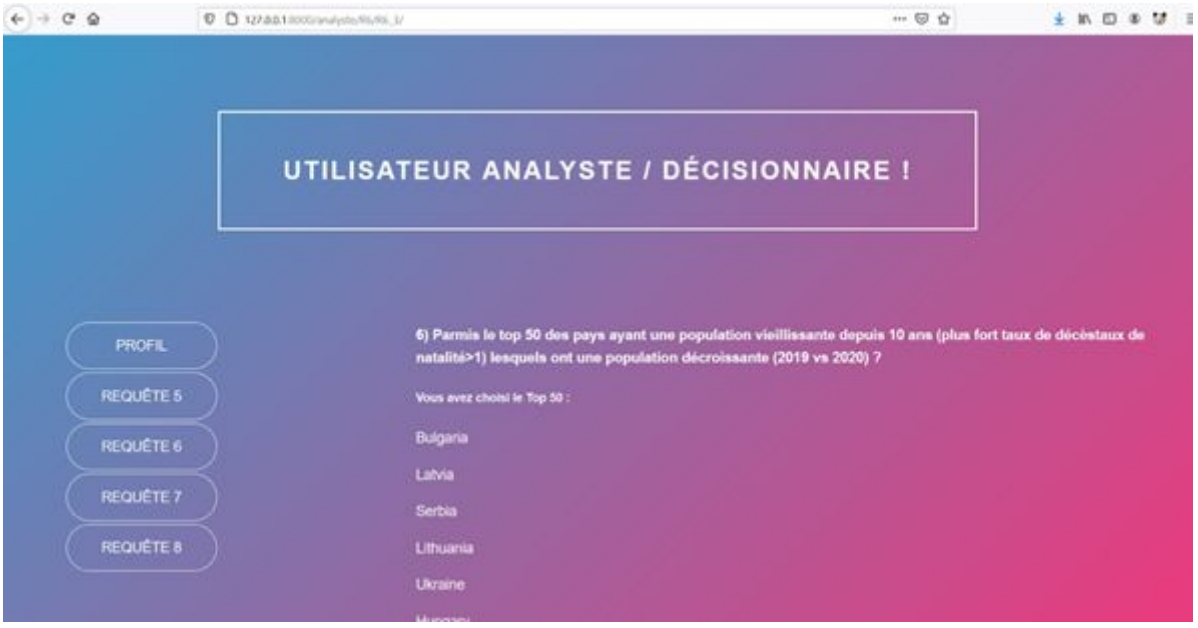
Choix du top pour 6)



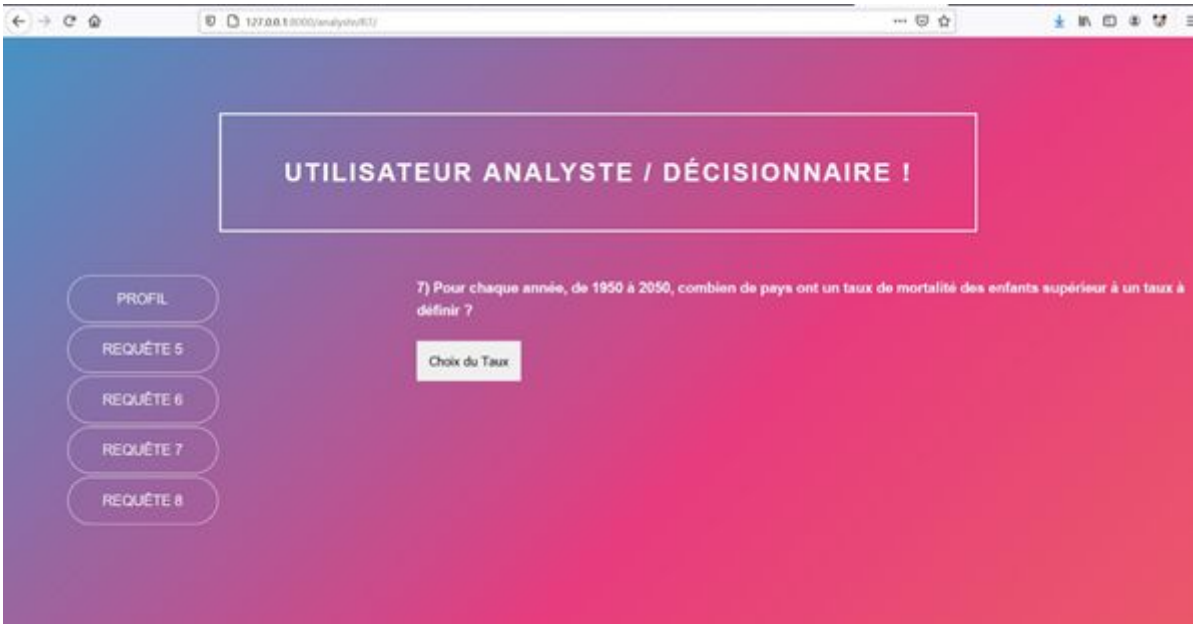
Choix du top 10 pour 6)



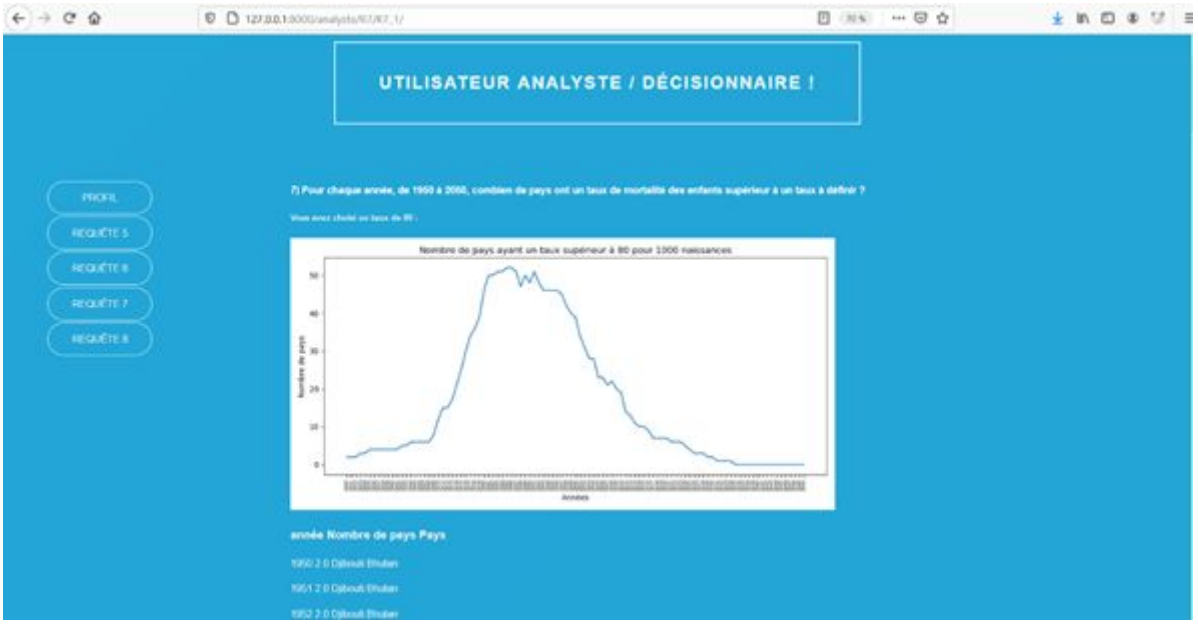
Choix du top 20 pour 6)



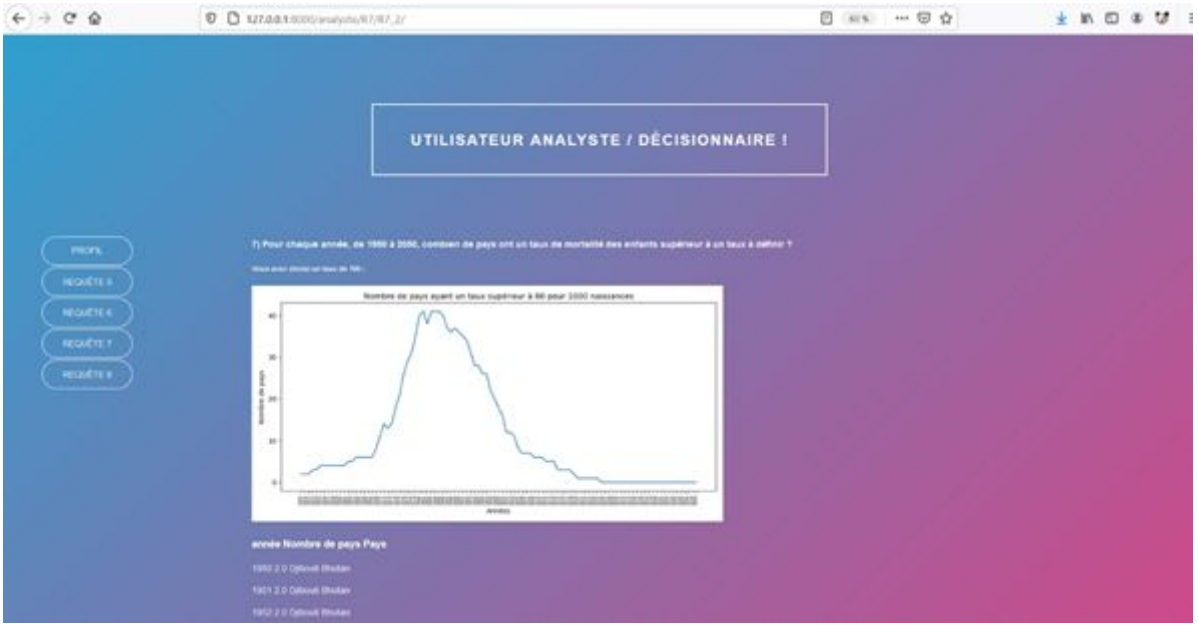
Choix du top 50 pour 6)



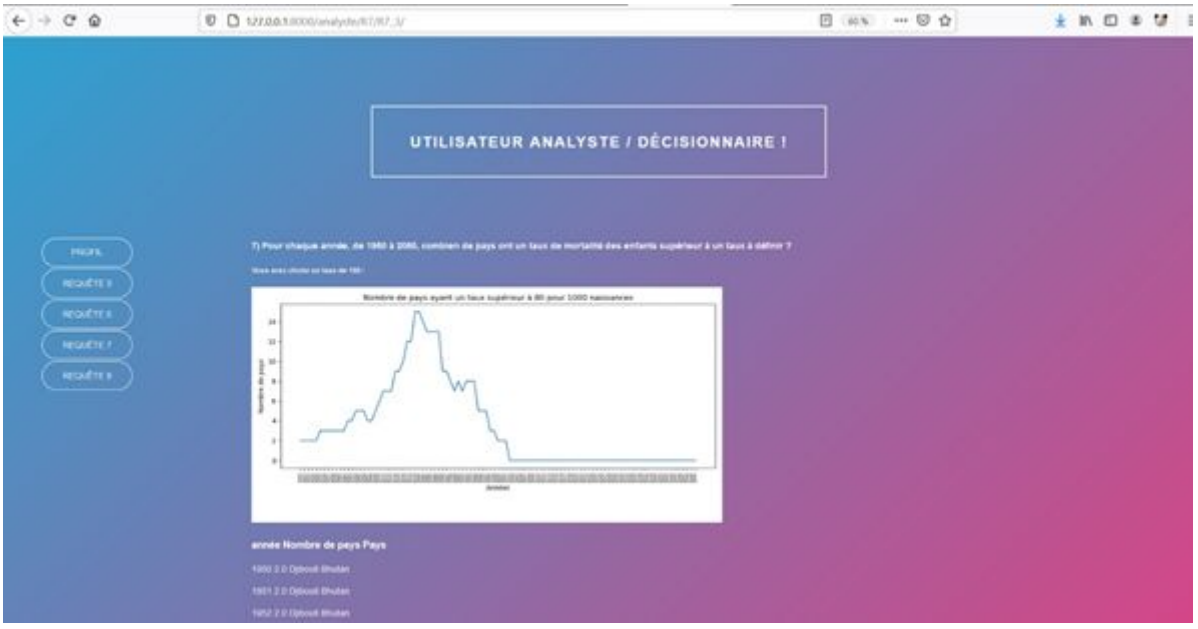
Choix du taux pour 7)



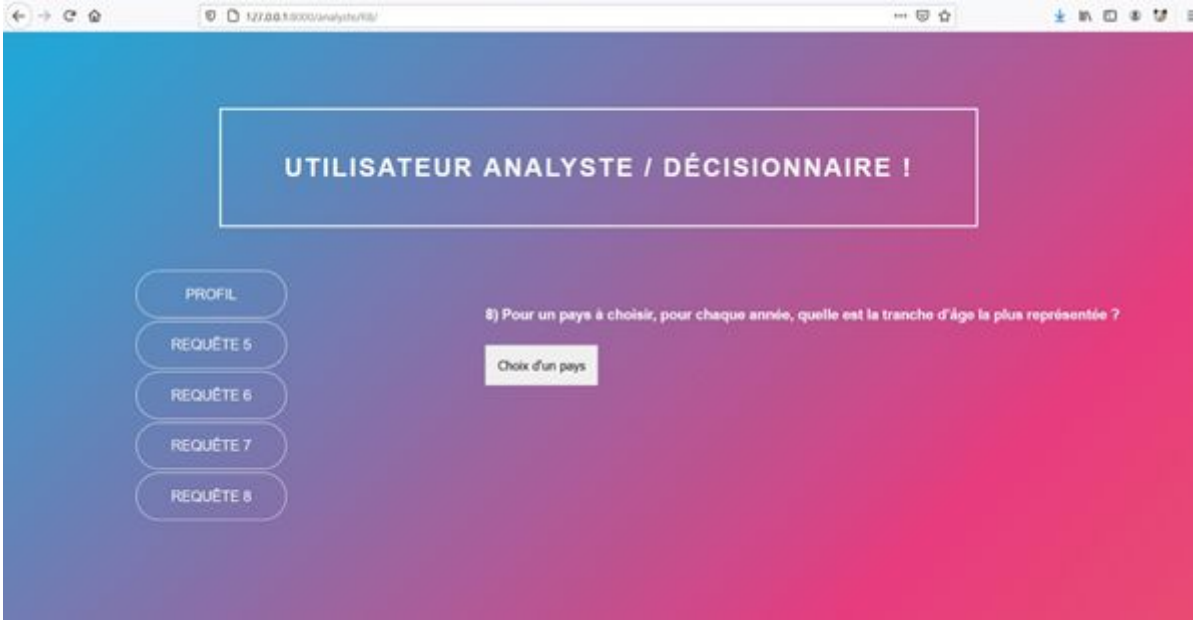
Choix d'un taux de 80 pour 7)



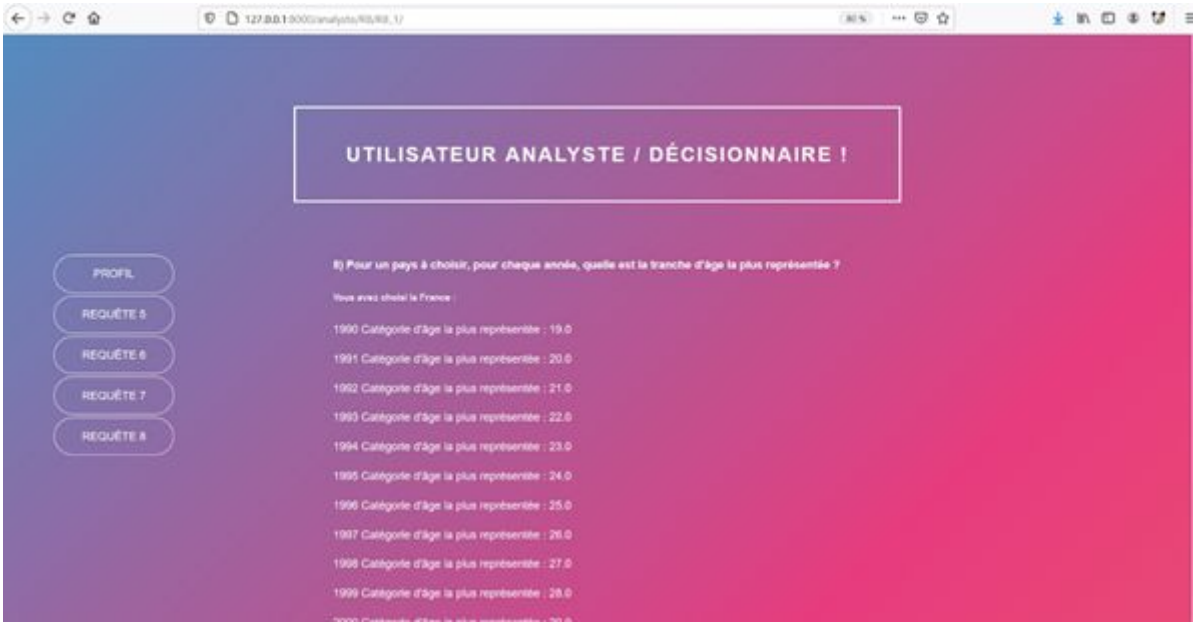
Choix d'un taux de 100 pour 7)



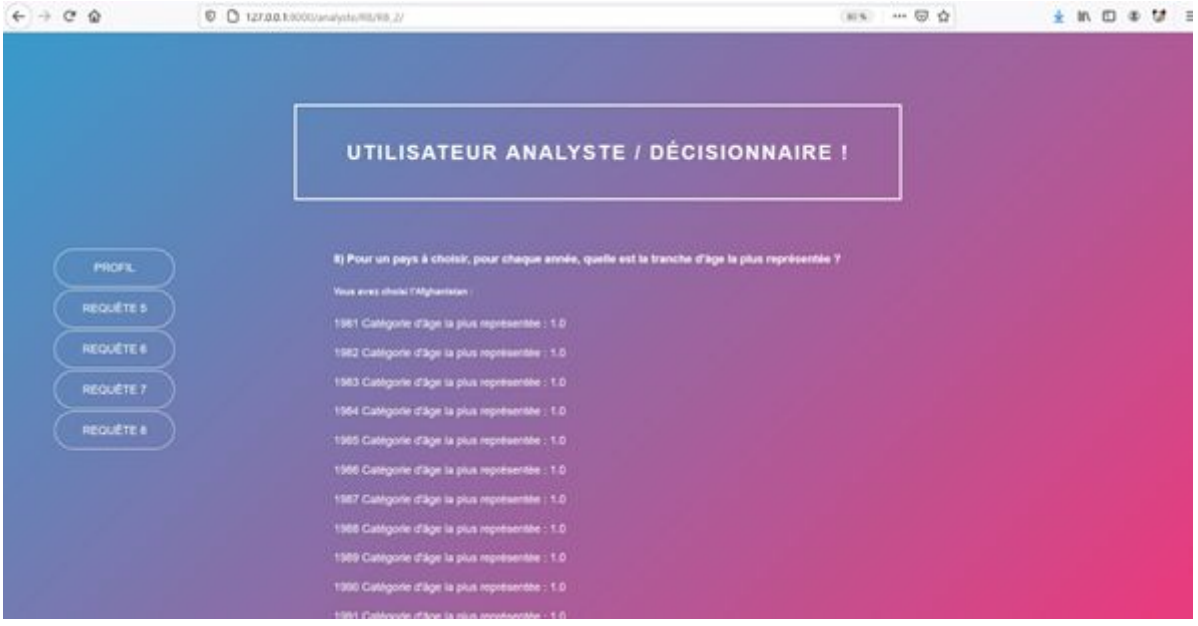
Choix d'un taux de 150 pour 7)



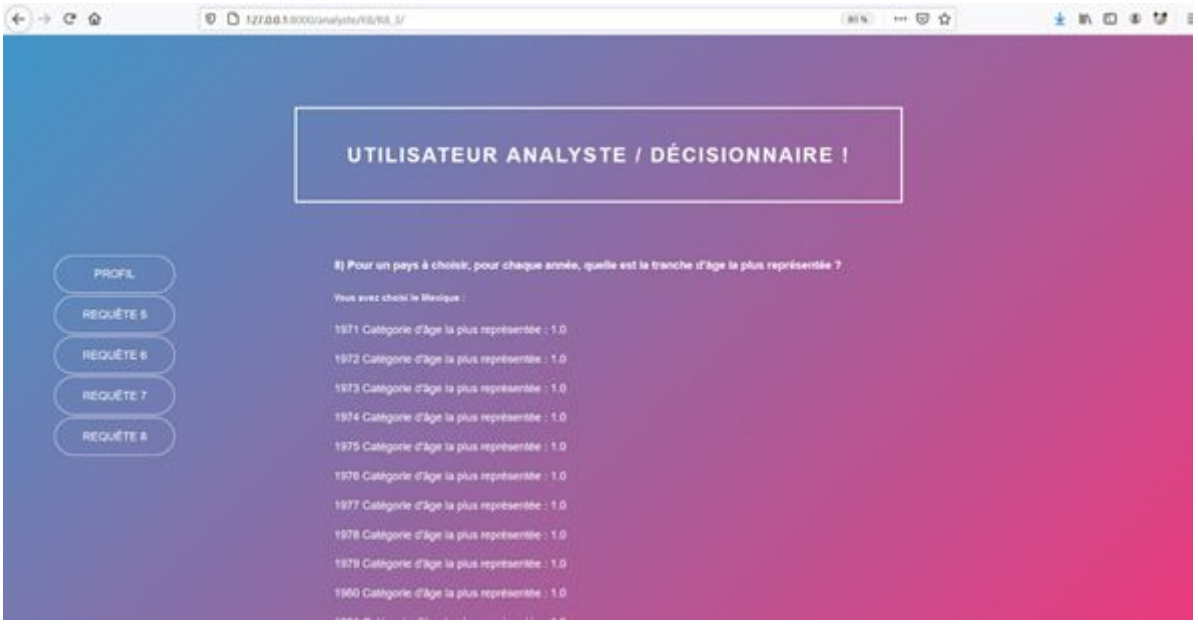
Choix du pays pour 8)



Choix de la France pour 8)



Choix de l'Afghanistan pour 8)



Choix du Mexique pour 8)

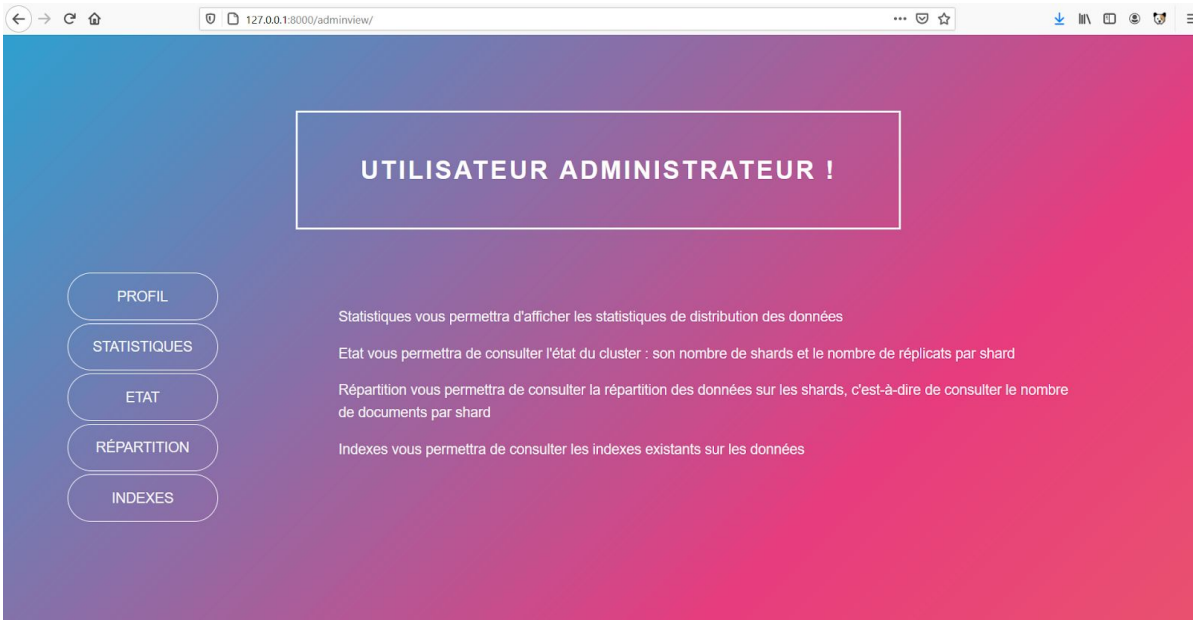
3. Administrateur

L'administrateur de la base MongoDB doit pouvoir récupérer différentes statistiques pour pouvoir faire évoluer le cluster en fonction de la charge. La vue doit pouvoir fournir les informations suivantes :

- Les statistiques de distribution des données ;
- Consulter l'état du cluster : nombre de shards, nombre de réplicats par shard
- Répartition des données sur les shards (nombre de documents)
- Indexes existants sur les données

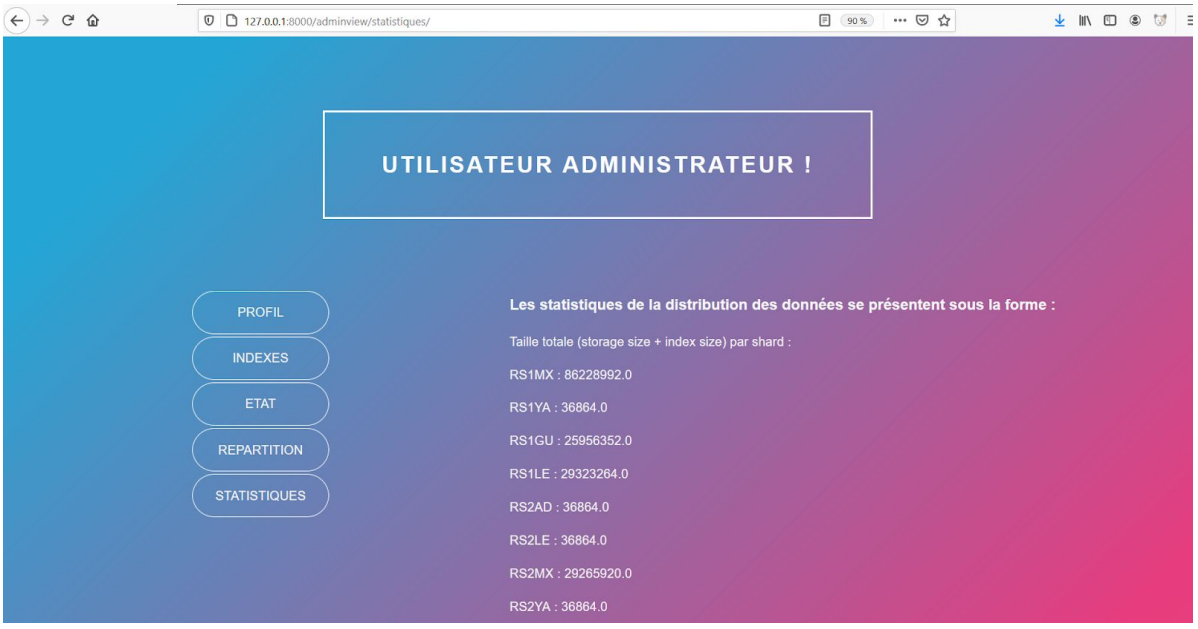
Après avoir fait le calcul de coûts , notre configuration est positionnée sur un seul shard . De ce fait l'ensemble des données est situé le shard RS1MX. Nous avons laissé cette configuration pour bien mettre en valeur la différence de répartition des documents dans le cluster pour la partie administrateur.

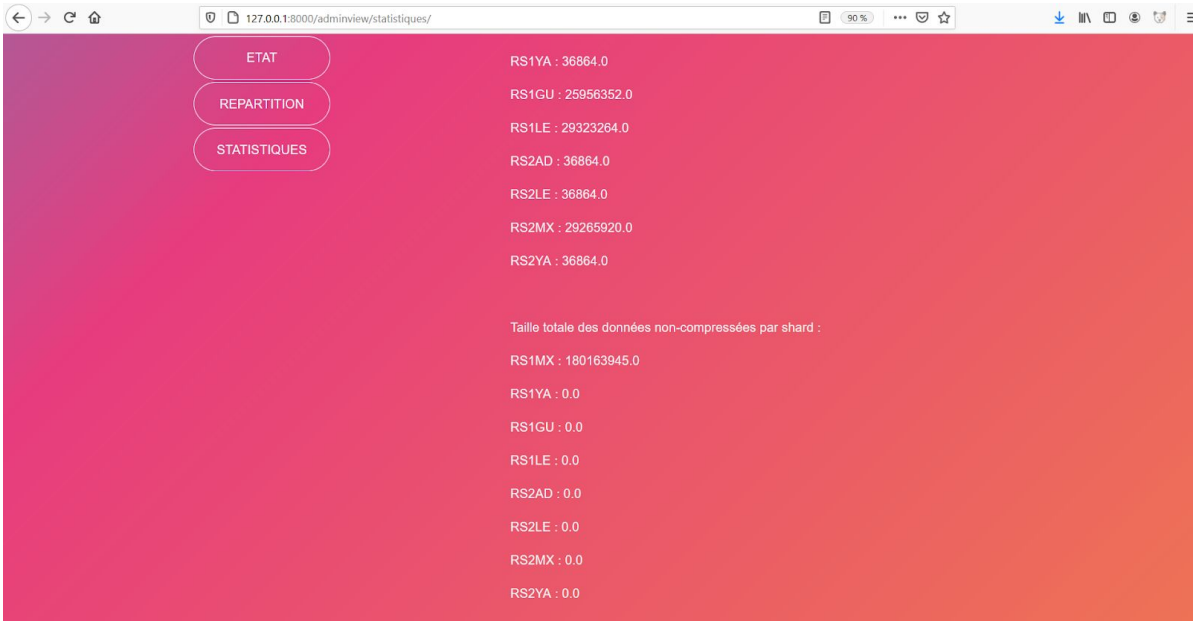
Pour revenir à une configuration distribué c'est-à-dire sur 6 shards, nous devons rajouté des shards via la commande mongo.

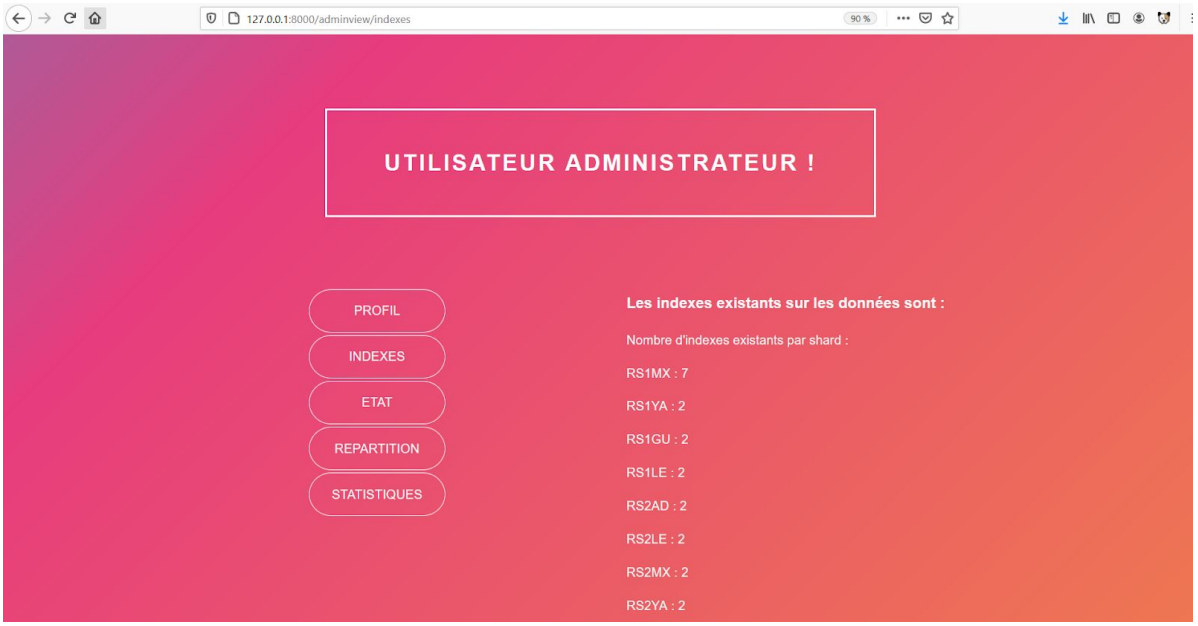


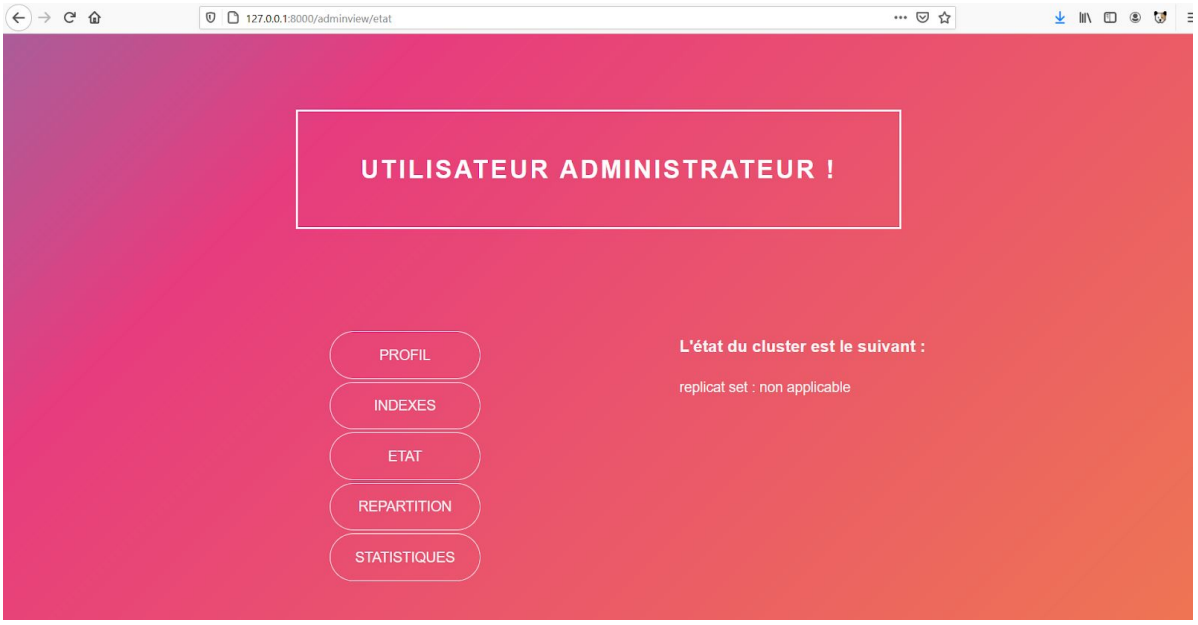
Arrivé sur la page administrateur, le menu propose encore de retourner au profil et d'accéder aux options suivantes :

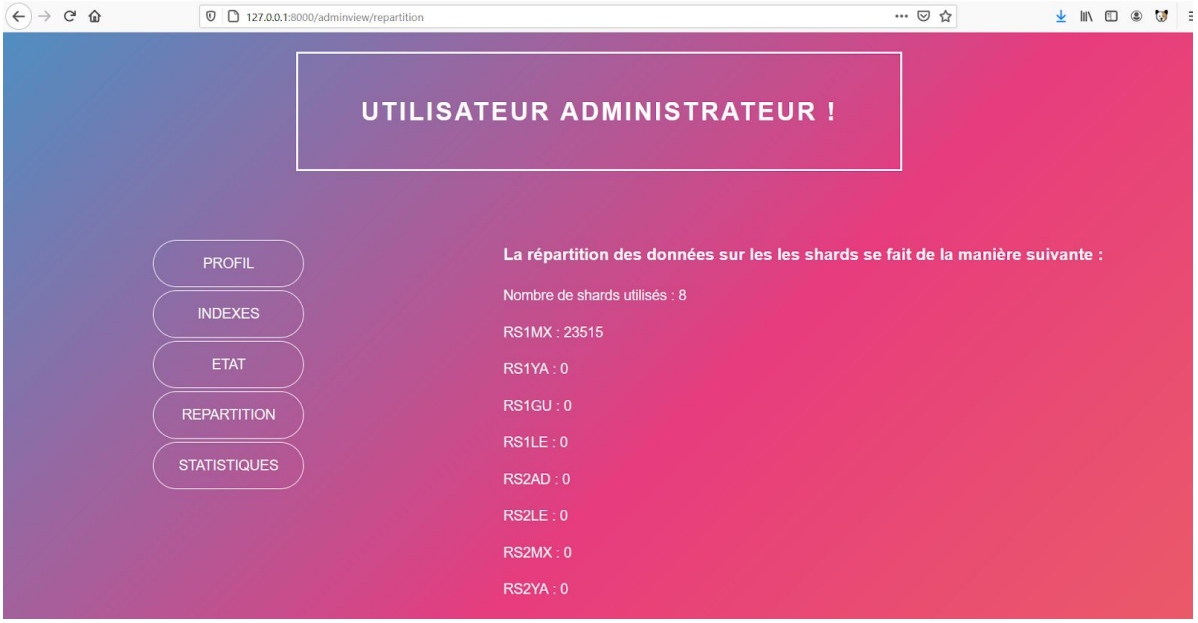
- Statistiques : consulter les statistiques de distribution
- Etat : voir l'état du cluster c'est à dire le nombre de shards, nous n'avons pas de réplicaset
- Répartition : répartition des données sur les shards (nombres de documents par shard)
- Indexes : voir les indexes sur les données


127.0.0.1:8000/adminview/statistiques/
90 %
UTILISATEUR ADMINISTRATEUR !
PROFIL
INDEXES
ETAT
REPARTITION
STATISTIQUES
Les statistiques de la distribution des données se présentent sous la forme :
Taille totale (storage size + index size) par shard :
RS1MX : 86228992.0
RS1YA : 36864.0
RS1GU : 25956352.0
RS1LE : 29323264.0
RS2AD : 36864.0
RS2LE : 36864.0
RS2MX : 29265920.0
RS2YA : 36864.0


127.0.0.1:8000/adminview/statistiques/
90 %
ETAT
REPARTITION
STATISTIQUES
RS1YA : 36864.0
RS1GU : 25956352.0
RS1LE : 29323264.0
RS2AD : 36864.0
RS2LE : 36864.0
RS2MX : 29265920.0
RS2YA : 36864.0
Taille totale des données non-compressées par shard :
RS1MX : 180163945.0
RS1YA : 0.0
RS1GU : 0.0
RS1LE : 0.0
RS2AD : 0.0
RS2LE : 0.0
RS2MX : 0.0
RS2YA : 0.0


127.0.0.1:8000/adminview/indexes
90 %
UTILISATEUR ADMINISTRATEUR !
PROFIL
INDEXES
ETAT
REPARTITION
STATISTIQUES
Les indexes existants sur les données sont :
Nombre d'indexes existants par shard :
RS1MX : 7
RS1YA : 2
RS1GU : 2
RS1LE : 2
RS2AD : 2
RS2LE : 2
RS2MX : 2
RS2YA : 2





V. Code

Le code de l'application est disponible sur Github.

Lien github : https://github.com/leyousse/DevAppCloud

Vous pouvez consulter une vidéo de l'application ici : https://drive.google.com/drive/folders/1AMVIATfN0rfaP_I8FkAdScsMrXZ-p0LS?usp=sharing