Project: Fast Migrate

COMP 3100

Professor: Dr. Amilcar Soares

Team 16

Ellodie Ha soon 201820248

Hans Ramasamy 201819943

Mohammad Hasan 201858685

(a) Introduction

Our main datasets originates from the World Happiness Report, which ranks countries by happiness levels from Gallup World Poll. Experts from different fields recognise that well-being is crucial in the progress of a nation. So, we shall apply these happiness indicators to develop a tool for informing people in decision-making such as migration. Another potential user is a student who is interested in studying abroad, but has difficulties in choosing a country. Who does not seek happiness? We believe that it plays a crucial role in deciding where to move. The dataset consists of scores and rankings of different countries based on answers to surveys from nationally representative samples, which then use Gallup weights to make estimates representative. The happiness score estimate is affected by the following factors: economic production, social support, life expectancy, freedom, absence of corruption and generosity.

We plan to build a dashboard and implement a world map with different colors corresponding to levels of happiness. Data visualization including bar charts, graphs and rankings shall be designed to help users in comparing and deciding between countries. An account feature is to be integrated to save users time from having to perform the same searches several times.

(b) Proposal

With the overwhelming number of possibilities that the world has to offer, immigration can be a challenging decision for people interested in settling in foreign countries. Fast Migrate is the web application our team will work on to bring relevant information of countries around the world all in one place. Whether it is global rankings, happiness scores, cost of living index, employment or crime rates among others, we want to ease the problem of looking all over the internet by saving users time and searches. Moreover, we will provide statistics with presentation of requested data to users interested in analyzing trends or contrasting between countries.

Unfortunately, our main dataset consists of happiness only and might not be sufficient for users looking for other criteria while evaluating their choice of destination. Hence, we consider getting access to a third-party API, which will complement our original dataset and provide the following factors: cost of living index, quality of life index, employment rates, crime rates, health care index.

Hence, our goal is to deliver a web application targeted to people seeking information about different countries, designed for, but not limited to immigration. Our team aims to achieve all the unique functionalities of Fast Migrate without compromising on aesthetics or website performance by the end of the third iteration.

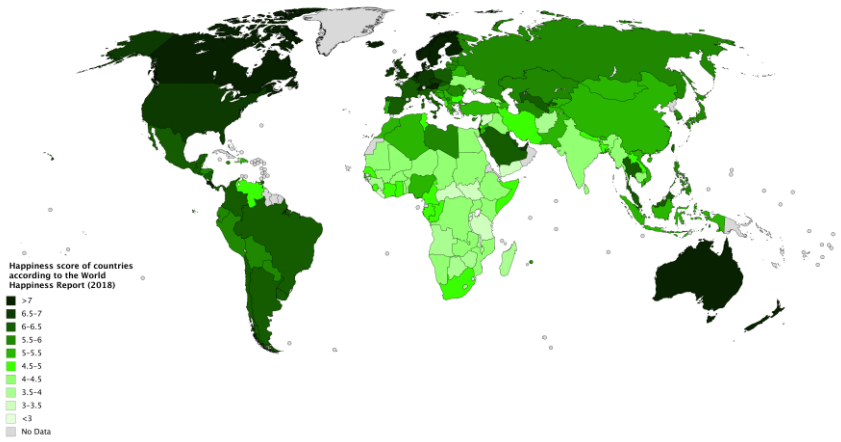
(c) Functionalities

- The user can get the information of a particular country he/she desires to immigrate to by inputting the name of the country.

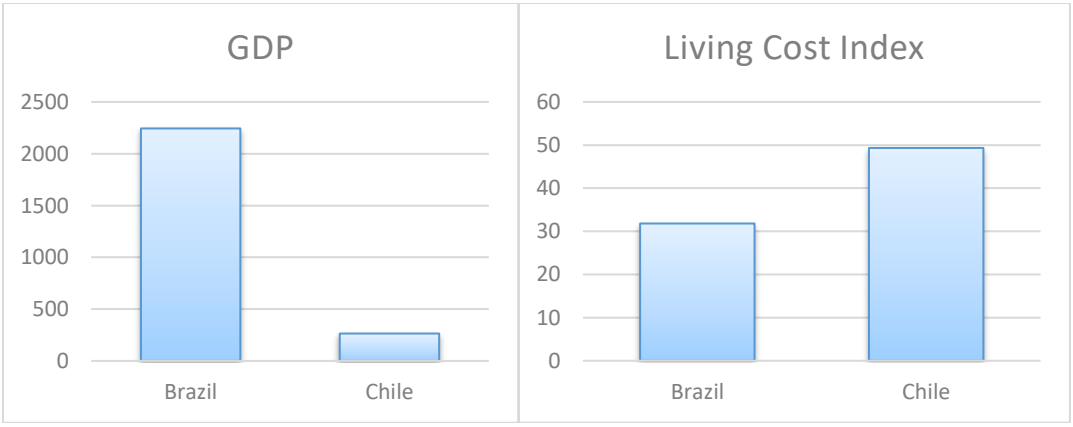
- The user can input a criterion he/she values most to get the rankings of countries accordingly, such as sorting by low crime rate or high employment level.

- The user can create an account, login and save its progress and search history.

- The user can get a colorized overview of all countries by a world map with each country having a specific color depending on its happiness level, shown below:



- The user can compare different factors (happiness, employment level, living cost, crime rate, GDP, etc.) of two countries of their choosing through charts, for example:



Iteration 2 - Documentation

1. Server-side of FastMigrate

The server-side of our application follows the MVC pattern which consists of model, view (routes) and controller. The model consists of data-related logic, that is, the transfer of data between the Routes and Controller components. For example, a user can retrieve country information from the database. The routes component is required for the user interface of the application, which will be assembled in the client-side of the project. The controller comprises interactions between the model and routes for requests processing and data manipulation. FastMigrate API was implemented using express to provide functionalities over HTTP requests.

Functionalities

At this point, we have implemented the functionalities required for the back-end of the application. For instance, a country's name is used as input to query the database and extract relevant information. The remaining functionalities are to be implemented when the front-end is assembled. The functionality of user account might be removed if the implementation of sessions and login might be too complex and beyond our knowledge. It is until the client-side has been implemented that we will be able to test and check if it works.

2. Models

Methods required for the manipulation of our datasets to/from mongoDB. They are necessary for the correct insertion, update and retrieval of countries' information.

Methods	Descriptions
_get_country_collection()	Connects to the proper collection
addCountryInfo(happiness_score, Gdp, unemployment_rate, crime_index, quality_of_life, health_care, cost_of_living)	Adds different factors of country to the database
isValid()	Checks if passed parameters match the required fields
save(db)	Saves country to the database
update(db, name, happiness_score, Gdp, unemployment_rate)	Updates information of a given country to the database
delete(db, name)	Deletes information about a given country from the database
getCountryByName(db, name)	Gets a country by name and returns all data of requested country from the database

3. Controllers

Methods
create
getOne
updateOne
deleteOne
all

Routes

Parameter name: name
Description: name of country (First letter capitalized)

```
// To access paths:
'http://localhost:3000/country'

// To add a new country to the database over HTTP request:
router.post('/', country_controller.create);

// To get the list of all countries
// from database over HTTP request:
router.get('/', country_controller.all);

// To get the information of a country from database
// by inputting name of country over HTTP request:
router.get('/:name', country_controller.getOne);

// To update the information of a country in database
// by inputting name of country over HTTP request:
router.put('/:name', country_controller.updateOne);

// To delete the information of a country from database
// by inputting name of country over HTTP request:
router.delete('/:name', country_controller.deleteOne);
```

Resource Information

Response formats JSON
Requires api_key for 3rd-party API (provided in code)

Example Requests:

GET
https://localhost:3000/country/Canada

Example Response:


```
{
  _id: 6046d4a462f2b3357cf762dd,
  name: 'Canada',
  happiness_score: 7.278,
  Gdp: 1794,
  unemployment_rate: 6.5,
  crime_index: 41.422305967278035,
  quality_of_life: 159.20378435706945,
  health_care_index: 71.71889142374837,
  cost_of_living: 69.25370113027466
}
```


Update:

POST https://localhost:3000/country/	Adds a country to the database
PUT https://localhost:3000/country/Brazil	Update the fields of Brazil in the database
DELETE https://localhost:3000/country/Canada	Deletes Canada from the database

4. Design of Data Model collection(s)

Our data is designed for usage pattern such that data model evolution is made easy. Hence, the data model collection follows the embedded pattern because there are small subdocuments (happiness.json, unemployment.json and the rest of the factors from the 3rd-party API) which do not grow by large amounts. Our data does not need to change regularly and faster reads are preferred for data visualization.

For a sneak peak of our sample data, please run ‘node populate_sample.js’ to populate the mongo database with 10 countries (**Note**: It is not required for the testing of our API).

5. Tests

Mocha is used for the testing of our server-side. Tests of simple and complex cases covering success and failure were performed on models.

Simple Cases:

Success 1 - Test creation of a valid country with parameters matching
Fail 1 - Test an invalid Country name
Fail 2 - Test an invalid Country happiness
Fail 3 - Test an invalid Country gdp
Fail 4 - Test an invalid Country unemployment
Success 2 - Test the insertion of a valid Country (Country.save)
Success 3 - Test the update of a valid Country (Country.update)
Success 4 - Test the deletion of a valid Country (Country.delete)
Success 5 - Test the retrieval of a Country by name (Country.getCountryByName)
Success 6 - Test the retrieval of all Countries (Country.getCountries)

Complex Cases:

Success 1 - POST /country, DELETE /country/name
Success 2 - POST /country, GET /country (retrieval greater than 1), DELETE /country/:name
Success 3 - POST /country, GET /country/:name, DELETE /country/:name
Success 4 - POST /country, PUT /country/:name, GET /country/:name, DELETE /country/:name

Update:

The mentioned tests were required to make sure the API works as designed under successful circumstances as well as unlikely scenarios. It is a way of validating different possibilities of manipulating the application. Hence, testing is a way of running checks for complete coverage.

Iteration 3 - Documentation

1. Client-side of Fast Migrate

The client-side involves interactivity and display of the data managed by our server-side. It consists of scripts embedded on the client's web page. Javascript is the language used to write the scripts which interact with HTML elements and CSS for styling and presentation of the content. jQuery alongside ajax calls were required for the integration of server and client sides.

Functionalities

At this stage, our team has worked on the implementation of most mentioned functionalities. A nav bar with three buttons was implemented to deliver the functionalities. Clicking on the 'World Map' button displays the colorized world map according to happiness levels. The 'Detail' button allows the user to input the name of the country and get its information. Rankings of countries and the bar charts for comparison are accessed by the 'Compare' button. Everything went according to plan with nothing new added. Unfortunately, the user account was the only functionality removed.

2. Views

Views refers to the 'public' directory of our code. Ajax calls are made to the server-side. Each script (.js) manages the components of the corresponding tab:

compare.js allows the user to input two countries for comparison and chooses a criterion of their choice from the drop-down menu. It then displays the bar chart comparing the two countries and a list of top 10 countries according to the field chosen.

Methods	Descriptions
getFieldFromRes(response, field_name)	Gets field selected from drop-down and returns that field from response
getRanking(response, field_name)	Gets field selected from drop-down and returns a sorted array of all countries based on that field

detail.js manages an event to the 'detail' button. After the user inputs the name of a country, it is identified in the database when the search button is clicked. Its corresponding fields are then displayed with animations.

Methods	Descriptions
fixedDeci(country_field)	Checks if a country field is missing so as to prevent errors from displaying country's information

main.js maintains the behavior of the tabs such that the world map, charts and information do not get overlapped one on the other.

world_map.js makes use of leaflet and map.geojson such that a world map of countries with respective borders is created. A layer is added to set different colors according to happiness scores. The darker the red, the less the happiness while a white coloring indicates that the information is not available for the country. 'legend' is created to show the colors associated with each happiness interval. When the 'world map' button is clicked, all the components are assembled. Clicking on each country will pop up its corresponding happiness score and by clicking on 'show more' displays the remaining fields of the country.

Methods	Descriptions
addHappiness()	Ajax call to query server-side to get all information of all countries and add the happiness data into the geoJson
getColor(happiness)	Decides which fading red color of corresponding happiness interval
style(features)	Sets style of country on map according to happiness level
showMoreDetails(name)	Ajax call to get information of selected country on map

3. Elements in assembling the client-side

Element	Explanation
html	Root element defining whole HTML document
body	Defines document's body
h1	Heading used for "Fast Migrate"
p	Paragraph for text
script	1 cdn for jquery, 1 for custom fonts, 1 for leaflet, 1 for chart & scripts for event handling
form	Gets input from user to query database (mongoDB)
button	Controls corresponding event when clicked
canvas	Displays bar chart

style.css is used to modify the way HTML elements are displayed such as different sizes, colors, layouts and fonts.

4. Animation effects

Hover effects: When the user mouse over the buttons, the background-color changes from white to blue.

Text-typing effects: After clicking on 'detail' button, when the user inputs a country name, the information is displayed as if the text was being typed.