COMP 3100 Project Iteration 2

Human Height Visualizer

Bailey Liang : 201623444
ZhiWei Fan : 201811171
Group 37

Introduction

For this project, we will be creating a data visualization system using the public Kaggle dataset 'Height of Male and Female by Country 2022' by 'Majyhain'. The dataset is a comma-separated-value (csv) file containing the average heights in both metric and imperial units for both males and females of each country. It contains a total of six (6) columns: the first of which correspond to the ranking of the country's average height in descending order with the tallest average being ranked first (1st), the name of the country, the average male height in cm, the average female height in cm, the average male height in feet and the average female height in feet.
The goal of this project is to create a visualization of the dataset allowing users to view the ranking of average human heights for each country, compare the average human heights of each country using human-shaped graphics corresponding to that country's average height and finally to compare the user's own height amongst the human heights in the dataset

Proposal

As stated in the introduction, the overall idea of this project is to create a visualization of human height by country for viewing and comparison purposes using the public Kaggle dataset mentioned. The default view will include the visualization of average human heights for each country using a male shaped graphic behind a contrasting female shaped graphic corresponding to the average male and female heights for that country respectively. The default view will attempt to visualize the different heights horizontally across the screen sorted by their ranking from tallest to shortest. The human shaped graphics will be selectable which highlights and freezes them on screen allowing the user to compare that country's average human height with the others by scrolling horizontally. Lastly the user will be able to enter their own height in either metric or imperial units which will create a new human graphic corresponding to the inputted height allowing the user to compare their own height with the countries of the world.

Functionalities

- Generate a mongodb collection from the specified dataset
- Generate statistics from this mongodb collection and storing the result in it's own mongodb collection
- Limit the number of entries to read from the mongodb dataset collection to determine number of human-shaped-graphics to draw
- Sort the entries read from the mongodb dataset collection according to parameters to determine sorting order of human-shaped-graphics
- Parse height data obtained from the entries to determine y-scale of each human-shaped-graphic

- Convert height data obtained from the entries into metric or imperial units to allow visualization of both units
- Toggle viewing only male heights, female heights or both male and female heights of each country
- Draw contrasting horizontal lines across all human-shaped-graphics corresponding to the tallest and shortest males and females in the dataset
- Selectable highlighting of human-shaped-graphic to freeze x-position on screen allowing comparison to other countries by scrolling horizontally
- Create custom entries in the dataset allowing custom user-defined heights to be compared amongst the dataset
- Delete custom entries from the dataset
- Generate percentile ranking of selected human-shaped-graphic amongst the dataset

Server-side Implementation

The server was implemented following MVC design principles in node.js using the express, mongodb and mocha modules. New functionalities implemented include generating the mongodb database collection from the dataset, generating statistics from the dataset and storing the results into its own mongodb collection. Basic CRUD operations are also implemented for both the `heights` and `height-stats` models allowing user-defined entries to be created, updated and deleted. The `heights` model also implements an additional method for limiting and sorting entries returned from the collection. In general, functionalities for this project iteration remain the same as before albeit re-written to be more specific and accurate with respect to the server implementation.

Models

Two models were implemented in this project, the `heights` model and the `height-stats` model as follows:

`./models/heights.js/`:

- the `heights` model is responsible for managing and interacting with the main dataset
- and includes the following methods:
    - `.resetDatabase()` : which generates the mongodb collection from the specified `dataset.csv` file found in the `./commons/` directory
    - `.createHeightObject()` : which creates a `height_object` given a name, male height, female height and an optional boolean specifying whether this object contains data for a country or is a user-defined entry
    - `.createRecord()` : which takes a given `height_object` as an argument and saves that `height_object` into the mongodb collection

        - `.readRecord()` : which takes a given name and attempts to return the `height_object` containing that name, as an array
        - `.readRecordEx()` : which takes a given search query object, sort query object and a number specifying the maximum number of entries to return from the collection; these methods are all optional and their default values returns all the entries in the collection
        - `.updateRecord()` : which takes a given name and `height_object` containing the updated record and attempts to update the entry with the given name inside the collection
        - `.deleteRecord()` : which takes a given name and attempts to remove the entry containing that name, from the collection

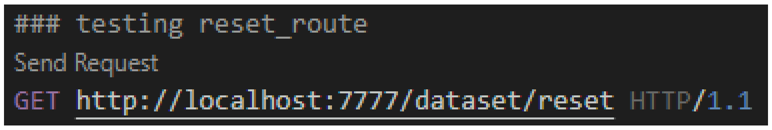
`./models/height-stats.js` :

- the `height-stats` model is responsible for handling and generating relevant statistical data from the dataset including user-defined entries
- and includes the following methods:
    - `.createStatObject()` : which creates a `stat_object` given the name of the statistic and its value
    - `.generateStatistics()` : which generates statistical data such as min, max, range, count, sum, mean, variance and standard deviation for both male and female heights in the dataset and saves this as their corresponding `stat_object` in the collection
    - `.createRecord()` : which takes a given `stat_object` and creates a record using that object in the collection
    - `.readRecord()` : which takes the name of the statistic and attempts to return a record containing that statistic to the user, as an array
    - `.updateRecord()` : which takes the name of the statistic and a `stat_object` containing the updated value and attempts to update the entry with that statistic in the collection
    - `.deleteRecord()` : which takes the name of the statistic and attempts to remove the entry containing that statistic, from the collection
    - `.getPercentile()` : which takes the `name` of the height to find and the `sex` specifying which column to calculate the percentile of

Routes and Controllers

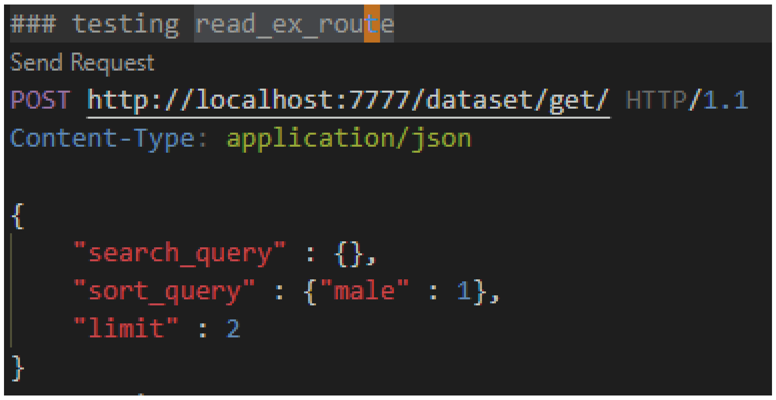
Two controllers are implemented to correspond with the /dataset/ path and /statistics/ path and they handle the following routes which can be tested using the Vscode extension `REST Client` by Huchao Mao along with the included `rest tester.rest`:

- **GET** `/dataset/reset/` : resets the database by clearing the collection and loading the original values from the dataset



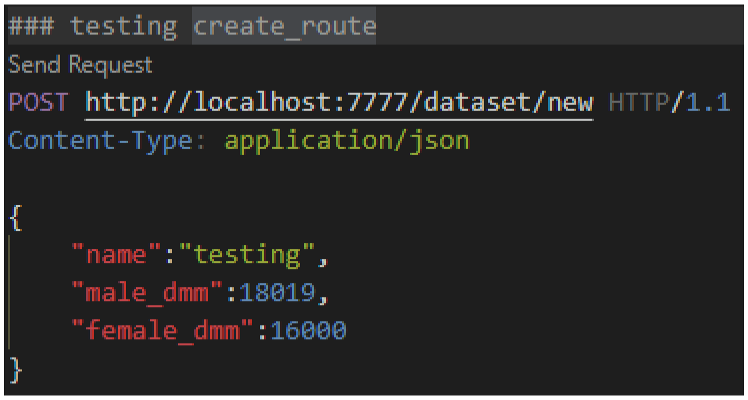
```
### testing reset_route
Send Request
GET http://localhost:7777/dataset/reset HTTP/1.1
```

- **POST** `/dataset/get/` : allows specifying a json containing advanced search queries to the database such as `search_query`, `sort_query` and `limit` with omitted values using default values instead



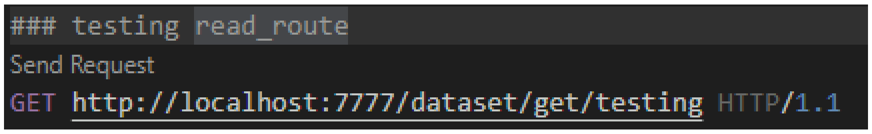
```
### testing read_ex_route
Send Request
POST http://localhost:7777/dataset/get/ HTTP/1.1
Content-Type: application/json

{
    "search_query" : {},
    "sort_query" : {"male" : 1},
    "limit" : 2
}
```

- **POST** `/dataset/new/` : allows creating new entries into the database by specifying a json containing `name` of the entry, `male_dmm` height and `female_dmm` height



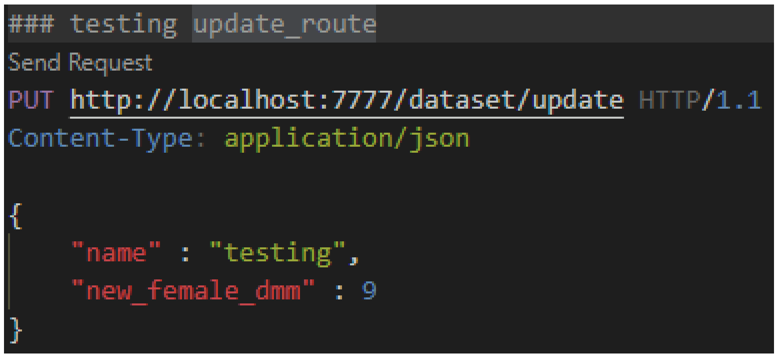
```
### testing create_route
Send Request
POST http://localhost:7777/dataset/new HTTP/1.1
Content-Type: application/json

{
    "name":"testing",
    "male_dmm":18019,
    "female_dmm":16000
}
```

- **GET** `/dataset/get/:name/` : reads a record from the database with the specified name

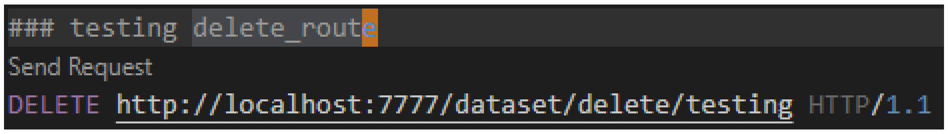


```
### testing read_route
Send Request
GET http://localhost:7777/dataset/get/testing HTTP/1.1
```

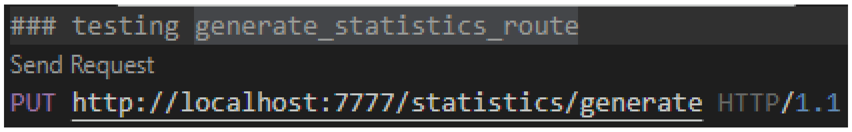
- **PUT** `/dataset/update/` : allows specifying a json containing the `name` of the record to update along with the `new_name`, `new_male_dmm` and `new_female_dmm` containing their updated values, if omitted will simply default to their original value in the record


```
### testing update_route
Send Request
PUT http://localhost:7777/dataset/update HTTP/1.1
Content-Type: application/json

{
    "name" : "testing",
    "new_female_dmm" : 9
}
```

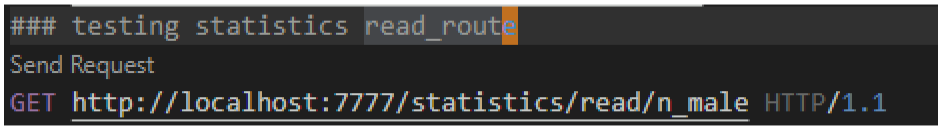
- **DELETE** `/dataset/delete/:name/` : deletes the record from the database with the specified name


```
### testing delete_route
Send Request
DELETE http://localhost:7777/dataset/delete/testing HTTP/1.1
```

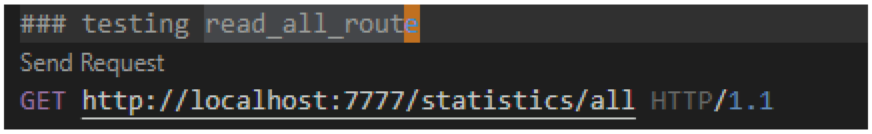
- **PUT** `/statistics/generate/` : generates statistical data for the dataset and store the result in its own collection


```
### testing generate_statistics_route
Send Request
PUT http://localhost:7777/statistics/generate HTTP/1.1
```

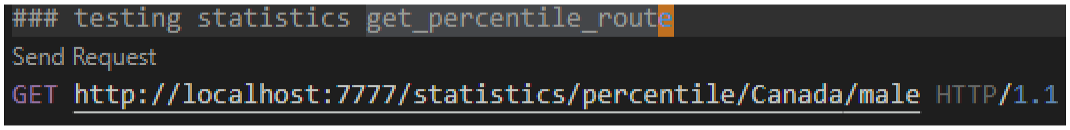
- **GET** `/statistics/read/:stat/` : returns the record containing the specified statistic


```
### testing statistics read_route
Send Request
GET http://localhost:7777/statistics/read/n_male HTTP/1.1
```

- **GET** `/statistics/all/` : returns all the statistics


```
### testing read_all_route
Send Request
GET http://localhost:7777/statistics/all HTTP/1.1
```

- **GET** `/statistics/percentile/:name/:sex/` : returns the percentile of the specified name and sex


```
### testing statistics get_percentile_route
Send Request
GET http://localhost:7777/statistics/percentile/Canada/male HTTP/1.1
```

although the `height-stats` model contains methods for creating, updating and deleting `stat_objects`, they aren't considered part of the primary use-case and routes enabling them are therefore omitted

Data Model
The normalized data model was chosen since we decided it would simplify the search queries to the main dataset collection if we didn't have to filter out the statistics object all the time.
The database is structured as two (2) collections corresponding to the main dataset and the statistics generated from the main dataset. Data in the main dataset is stored as JSON objects containing the `name`, `male` height, `female` height and a boolean determining whether the record `is_country`.
Data in the statistics dataset are stored as JSON objects containing the name of the `stat` and the `value` of the statistic.
Heights obtained from the dataset and statistical data generated are all recorded in decimilli-meters (dmm) (10^-4) initially implemented as a workaround to avoid passing floats through the URL but also has the benefit of minimizing potentially inaccurate float math in Javascript since calculations are mostly done with integer values and also because we think the integer values are more neat and easy to work with.

Tests
Testing was done using the `mocha` and `request` modules available from npm. `mocha` was used as the testing environment while `request` was used to make api calls inside the testing environment. Testing covers the most important and common use cases for the api, specifically the `heights` model and all the available api calls which mostly call functions from the `heights` model. The `.createHeightObject()` method was tested to be robust such that `height_objects` generated from this function were valid since they form the backbone of the project. All the api calls were then tested for the most common use cases with a few of them including additional tests for error checking and validation such as the Create, ReadEx, and Update calls since the Create and Update calls directly modify the database while the ReadEx call is versatile and should work as expected in multiple use cases.

References

- Kaggle Dataset of Human Heights, Height of Male and Female by Country 2022 | Kaggle
- Node.JS, Node.js (nodejs.org)
- Express, Express - Node.js web application framework (expressjs.com)
- MongoDB, MongoDB: The Application Data Platform | MongoDB
- Mocha, Mocha - the fun, simple, flexible JavaScript test framework (mochajs.org)
- Validator, validatorjs/validator.js: String validation (github.com)
- Request, request - npm (npmjs.com)