TECHNICAL TEST

Appscore**Ancestry**

I. BACKGROUND[1]

Appscore**Ancestry** is a *fictional* service offered by **Appscore Digital**. By collecting ancestral data from government census data with more than 100,000 genealogical records from across the country, the service is Melbourne's largest online family history resource.

The service would like to have a public web portal as an easy way for user to make genealogy search. Appscore Agency would like to ask for a talent programmer, you, to come up with a working prototype of the system.

II. INFRASTRUCTURE

- The service is powered by .NET Framework platform, using ASP.NET MVC framework.

- As being a prototype, the UI doesn't need to be fancy. It can be done using basic controls without paying too much attention into the design and layout (refer to the mocked up wireframe). The use of a front-end Javascript framework is not required but is recommended to complete the tasks.

- The database is provided in a JSON file with the following structure:

```
{
  "places": [
    {
      "id": 0,
      "name": "City of Melbourne"
    },
    {
      "id": 1,
      "name": "Carlton 3053 "
    },
    ...
  ],
```

[1]

```
  "people": [
    {
      "id": 100000,
      "name": "Theresa Chandal",
      "gender": "F",
      "father_id": 47912,
      "mother_id": 78460,
      "place_id": 13,
      "level": 67
    },
    {
      "id": 100001,
      "name": "Jodie Chandal",
      "gender": "F",
      "father_id": 47912,
      "mother_id": 78460,
      "place_id": 287,
      "level": 67
    },
    [...]
  ]
}
```

- The database consists of two types of entities:

Places. Each place has an **id** and a **name**.

People. Each person has an **id**, a **name**, their **gender**, **reference to their father**, **reference to their mother**, **reference to their place** and their **hierarchy level**.

Associations:

`person.father_id` self-reference to `person`
`person.mother_id` self-reference to `person`
`person.place_id` self-reference to `place.id`

IMPLEMENTATION TASKS

There are 2 JSON files created for you. **“data_small.json”** is filled with **~ 50** people and **~ 1,000** places and **“data_large.json”** is filled with ~ **100,000** people and the same amount of places.

You should use **data_small.json** to test for correctness of your logic, then use **data_large.json** to test your performance and algorithm design.

WHAT WE ARE LOOKING FOR

The purpose of this test is to evaluate the candidate on the following aspects:

- Programming in C# and ASP.NET.

- Problem solving.

- Code scaffolding and project structuring.

- Understanding of front-end web development with HTML5, JavaScript and CSS.

- API construction and consumption.

- Ability to understand the problem.

- Fluency in using **Git version control**. Please commit as you go so that we can see your progress. Please treat version control as if it is a normal project.

HOUSEKEEPING

1. Initialising an empty a git repository. During the course of your development, please commit frequently at appropriate checkpoints so that Appscore can evaluate the progress.

Please note that ****Submission of repository with only a single commit of the end result will result in a failure immediately****.

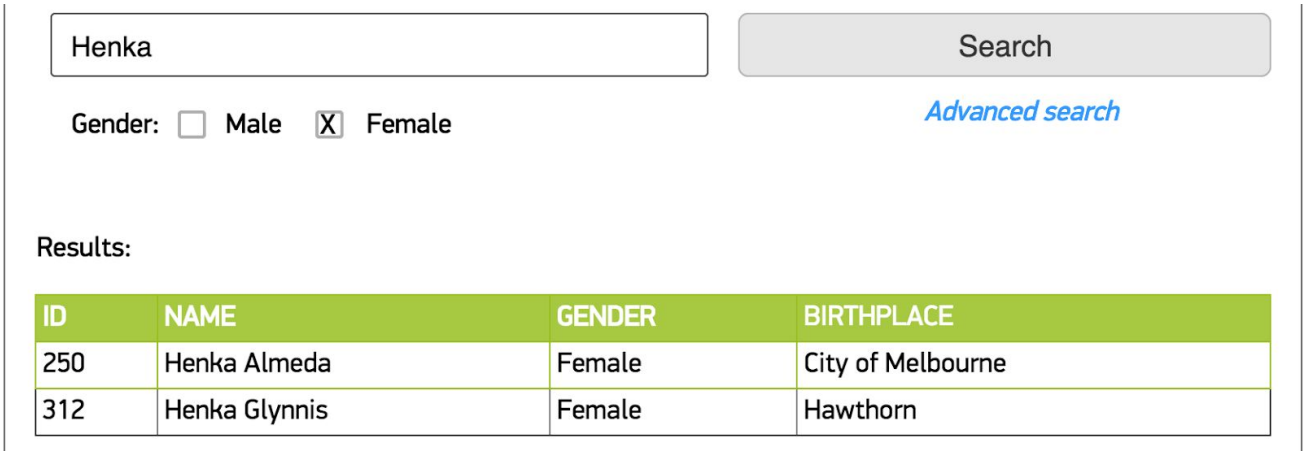
2. Scaffolding your project structure. Please feel free to use any framework and/or starter kit that you are comfortable with. Remember to place proper files and directory in your .gitignore file.

3. Push your git repository to a remote upstream, so that later you can provide Appscore access to your repository. You can utilise GitHub, GitLab, BitBucket or any other Git provider you are comfortable with.

TASK 1. Create the UI for the first page:

PAGE ONE: Home page / Simple search page

Our UX designer has come up with the mock-up of this page:



(please note the above results are for presentation only)

FIGURE 1. AppscoreAncestry homepage / Simple search page

It consists of:

1. One “Name” text box for the user to enter search keyword.
2. Two checkboxes whose labels are “Male” and “Female”.
3. A button, “Search”.
4. An advanced search link to link to advanced search page.
5. A results table.

TASK 2. Implement the business functionality for the simple search page so that the following workflow would success:

1. User type in search keyword in “Name” textbox (required field)
2. User select either “Male” or “Female” or Both or Nothing (having both or nothing selected result in both genders being returned)

3. User hits search
4. System retrieves a maximum of 10 records of people:
    + Whose name ***contains*** the keyword that the user search for.
    + *And* gender matches what the user selected
        (Male -> Only male records are looked up
        Female -> Only female records are looked up
        Both or nothing -> Both male and female records are looked up)

5. System display the result in a grid *(or in any other representation you prefer / think would better suit the scenario)*, on the same page.

***(Optionally)*: For larger data, instead of limiting the number of results to 10, implement pagination so that user can retrieve all relevant results.

6. User may choose “Advanced search” to navigate to the advanced search page

***(OPTIONAL) TASK 3*.** Create the UI and business functionality for the advanced search page:

PAGE TWO: Advanced search page

Our designer has come up with the mock-up of this page:

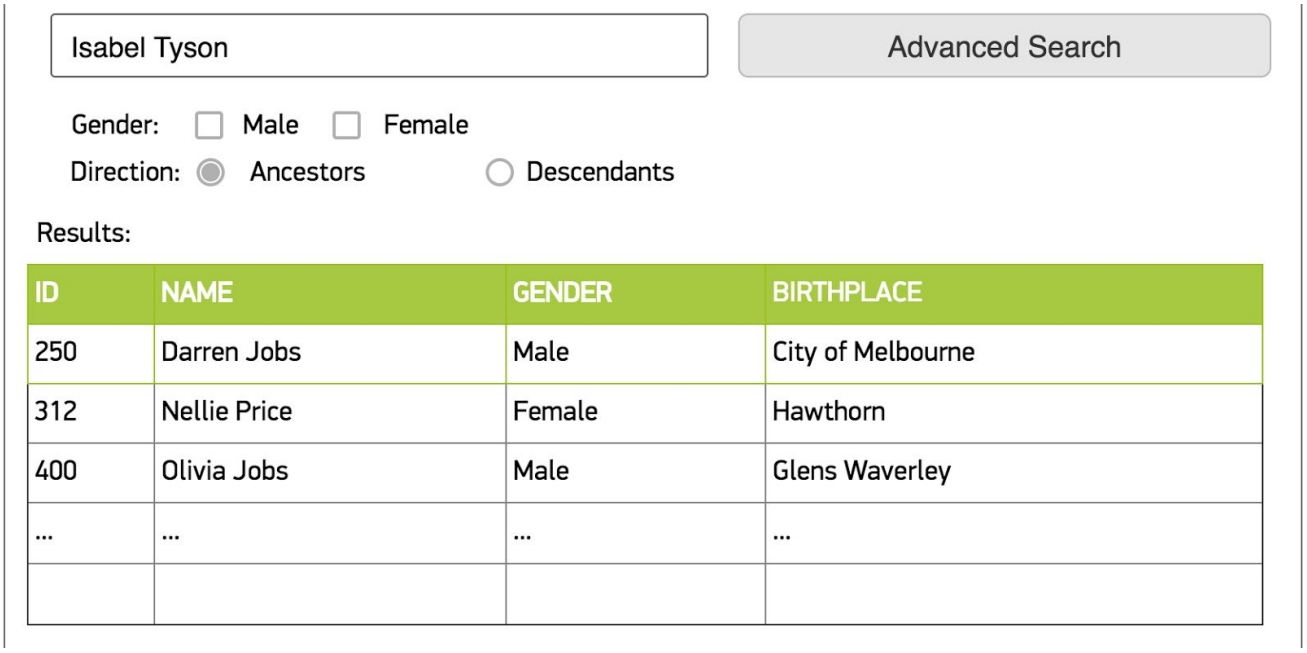


***FIGURE 2. AppscoreAncestry advanced search page** (with results)*

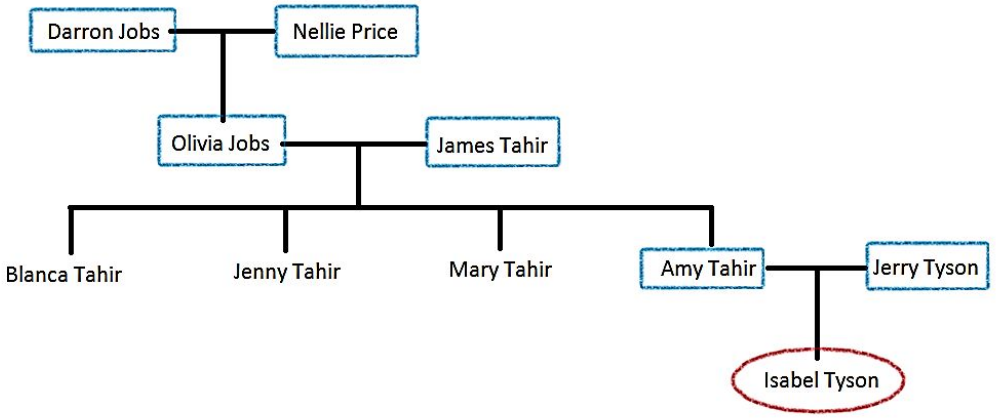
Similarly to the simple search page, the advanced search page has the name textbox, the gender options. However, this page give the user **the ability to search for a person’s Ancestors / Descendants**

1. The user select whether he/she wants to search for the ancestors or descendants of a person and type in search keyword
2. To make it easy, the system only finds the person whose name matches ***exactly*** (case-insensitive) and has the selected gender in the database. If there are more than one people satisfies the criteria, match the first person.
3. From the exists record, find a maximum of **10** *ancestors* or *descendants* of the person.

EXAMPLE

+ Consider the user searches for all **ancestors** of “Isabel Tyson”

+ If we have the following family tree:



+ The system should displays “Amy Tahir” (level 1), “Jerry Tyson” (level 1), “Olivia Jobs” (level 2), “James Tahir” (level 2), “Darren Jobs” (level 3), and “Nellies Price” (level 3) (level in this context means how far the ancestor is in relation to the searched person)

Another example with the above tree. If the user searches for all descendants of “Darron Jobs” and doesn’t care about genders we have to display “Olivia Jobs” (level 1), “Blanca Tahir” (level 2), “Jenny Tahir” (level 2), “Mary Tahir” (level 2), “Amy Tahir” (level 2) and “Isabel Tyson” (level 3) *(please note, for descendants search, spouses are ignored)*

** All names are randomly generated and are not associated with any real person.*