Datto Boston Practical

We include technical challenges in the recruiting process at Datto because one aspect of what makes an engineer successful here is technical competency. We are also strong believers that it takes more than technical expertise to make a great engineer, and we pride ourselves on making sure that challenges like this one are only part of the evaluation criteria. Other characteristics include cultural fit and a demonstrated desire and ability to take risk and continually learn from your mistakes.

That said, technical ability is one important characteristic, and this practical is useful to us in seeing more than just whether or not you can solve the problems, but additionally your style of coding and presenting your solution which differs widely from person to person.

We've carefully selected these problems to provide you with maximum flexibility and minimum time commitment, while still making sure the exercise provides meaningful information. For this reason:

● we provide you with a choice of problems to complete,

● the problems are not designed to be brainteasers or to be especially difficult,

● we want you to be able to code in a development environment that is comfortable for you, and

● the expectation is that they can be completed in just a few hours.

Input / Output Expectations: The means by which your program accepts queries and returns responses is up to you. Examples would be accepting requests through STDIN or command line argument and returning responses via STDOUT; or accepting requests via HTTP request and returning responses via HTTP response.

Timing Expectations: You may take as much time as you like on the selected task, but we don't expect you to spend more than 2-3 hours. If you find you've spent more than 2-3 hours, it is fine to wrap things up and send whatever you have; simply send us a note with the things you'd have liked to have done if you had more time.

Quality Expectations: Our only expectation is that the code works to the simple specification above, and that you are proud of the work product as something you'd be willing to ship. Use the language of your choice, and make use of any external libraries and resources you feel are appropriate.

Support: If you have questions about these tasks or the "specs" then feel free to contact Tyler Sakats directly at tsakats@datto.com.

Results: Please choose one of the tasks on the following page and submit all code and instructions needed to run the resulting application. This may be in any form you'd like that allows us to test and allows us to read the underlying code. You may send your solution via email through your recruiting contact or directly to tsakats@datto.com.

Option 1: CSV Load, Sort, Report

Use the language of your choice to write a program that will read in data from a CSV file and then support query operations on the people contained in the data set. Each row in the CSV represents a person, and the CSV contains the columns: id, first_name, last_name, age, github_account, date_of_third_grade_graduation. The id field will always be populated, but others are optional. The two operations your program should support are as follows:

1. Given a last name, return a list of all person ids with that last name.
2. Return a list of all people in the dataset, sorted by age. People with no age present in their data should sort to the end.

The program should support repeated query operations, but should read in data from the CSV file only once at program start-up. Please consider the performance of both query operations, and how they would perform for large data sets.

You can download a sample CSV file here.

Option 2 : HTTP Data Retrieval

Familiarize yourself with the subway API at http://myttc.ca/developers. Use the language of your choice to write a program that will use this API to do the following:

On startup, the program should allow the user to enter a location (e.g. 'finch_station'). For that location, the user should be able to make the following queries repeatedly:

1. Return the names and URIs of all stops at that location in alphabetical order by name. For example, the location “finch_station" would return the stop with name "Eastbound on Drewry at Gardenia Court" and URI “eastbound_on_drewry_at_gardenia_court", among others.
2. Given a stop name, return the names of all routes that stop at that stop (for the chosen location). For example, the location "finch_station" and the stop name "Eastbound on Drewry at Gardenia Court", it should return the route named "125 Drewry".

The program should support repeated query operations, but should make only one request to the TTC API to retrieve the information for the provided location. Please consider the performance of both query operations, and how they would perform for large data sets.