Warmup Collaborative Project

CS 3050 Fall 2024
Due Wednesday, Sept. 25th, 11:59 pm

1 Introduction

Teams of students will work together to create a command-line interface to query data that is stored in a Google Firebase Cloud Firestore database, a cloud-based document-model datastore. As a team, you'll write two programs that share much infrastructure: an admin program to load the data, and a query program to query the data.

For most of you, the concepts of a cloud-based document-model datastore will be new. You'll work as a team to learn what you need and then create the system. Some of you might be familiar with relational databases, and some of the general ideas from relational databases will be relevant. But the document model as a framework is also fundamentally different: the structural requirements are much looser, and there is no query-language interface (such as SQL).

You may use github or UVM's gitlab (https://gitlab.uvm.edu) for your code.

1.1 High-level description

Your system will allow a user to make queries, using a command-line interface, to data that's has been uploaded to a cloud datastore. Each team will select its data and develop its own data-specific query language.

1.2 Example

Suppose we select data representing local restaurants. Each restaurant entry will have several attributes: name, address, city, style, and cost (a value from 1 to 5). Each entry might also have an optional "accepts reservations" attribute, for which the value would be "yes" or "no".

As a user of this system, a realistic query that I might make would be to obtain a list of restaurants in Burlington. In the language of my system, this might have the following form:

```
> city == Burlington
```

and the response the system provides might be:

```
El Cortijo, Leunigs Bistro, Mr Mikes Pizza, Henrys Diner, Ali Babas Kabob Shop
```

(The initial `>` is a prompt that the query program prints to show that it is waiting for a query command from the user; you can use whatever prompt character(s) you like.)

Another query might be to find the restaurants that have a cost of 3 or less:

```
> cost <= 3
```

for which the response might be:

```
El Cortijo, Mr Mikes Pizza, Henrys Diner, Ali Babas Kabob Shop
```

The user should also be able to make compound queries. For example:

```
> city == Burlington and cost > 3
Leunigs Bistro, Single Pebble, Honey Road
```

Here are two important observations:

- the query language will be simple
- the query language will be specific to the data you've selected and put in your datastore

The language should enable intuitive queries but should be easy to parse (this is not ChatGPT).

2 The Programs

You'll create two programs: the admin program and the query program.

Use Google's Python interface to Firebase (see References below). You'll need to install the python `firebase_admin` module.

2.1 The admin program

The admin program will read data from a JSON file saved locally and will initialize and upload the data to a Google Firebase Cloud Datastore (not a Firebase Realtime Database). You'll run this program one time (if you run it a second time, it should delete and recreate the datastore).

2.2 The query program

The second program is the user query program—the commands shown above represent interactions with the query program. A user can run this program as often as they want. Each time the program starts, a new session starts. All requests for information take the form of a query in the "query language" of the system. The flow for the user is:

1. start the query program
2. make a query and receive a response
3. make additional queries, if desired, and receive responses
4. exit the program

To summarize: the data is uploaded once; after it's been uploaded it can be queried repeatedly.

3 Google Cloud Firestore

The data you will create will live "in the cloud", in a Google Cloud Firestore (not a Firebase Realtime Database) on Google servers.

3.1 Structure

Each item of data will be stored in a *document*. A document represents one data instance and has attributes that you define. It's roughly analogous to a row of data in a table in a relational database (RDB). Documents are grouped into a *collection*. A collection is roughly analogous to a table in an RDB.

For example, my project consists of restaurant data, which I store in a `Restaurants` collection. This collection contains documents. Each document represents a single restaurant and has the following fields: `name, address, city, style, cost, uuid`. Some documents have additional data a `reservations` field/ A field is roughly analogous to a column in an RDB table.

However, there are significant differences between an RDB and a Firebase datastore.

In a Firebase datastore:

- a particular field, and its data, might or might not be present in a given document
- fields don't have an enforced type, unlike in an RDB, although it's good practice to enforce consistency across documents; for example: in my project, `name, address, city, style` and `uuid` are strings, and `cost` is an integer
- collections are generally hierarchical rather than flat; each document in my top-level `Restaurant` collection can be treated as a tree, with the optional `Reviews` subcollection nested underneath it

Each document in a collection must have a unique identifier. You can either create that identifier yourself (which I do, with my `uuid` field); or you can ask Firebase to create a unique identifier for you when you add a document to a collection.

It's not necessary to use subcollections for this project.

4 The Data

Obtain (or create) some data suitable for saving in this format. Specifically:

- there should be a single kind of entity, for your top-level collection
- each entity should consist of some data that you expect to be present and some optional data that might or might not be present
- each entity should have at least three fields that will always be present
- each entity should have at least one optional field
- there should be at least 30 top-level instances of your entity

Examples of suitable datasets: musical artists, songs; countries or cities; athletes.

Many students have modeled Pokemon characters in the warmup project.

4.1 File format

Put your data into a JSON file. Your admin program will load data from your JSON file into your datastore.

I've put an example JSON file (from my query program) in gitlab: here, under `CS3050/Examples`.

4.2 Interface

Your admin program should take a single command-line argument: the name of the JSON file containing the data to load.

Your query program will enable a user to query the data using a simple command-line interface. For example, here's what the interface to my restaurant data might look like:

```
> city == Burlington
El Cortijo, Leunigs Bistro, Mr Mikes Pizza, Henrys Diner, Ali Babas Kabob Shop
> city == "Burlington" and cost < 4
El Cortijo, Mr Mikes Pizza, Henrys Diner, Ali Babas Kabob Shop
> cost "El Cortijo"
$$
> city == Burlington and stars > 3
Lounges Bistro
> stars > 4
no information
> stars > 3
Leunigs Bistro 115 Church Street Burlington, VT 05401 American 4 (reservations)
Als French Frys 1251 Williston Road South Burlington, VT 05403 American 1
Hinesburgh Public House 10516 Route 116 Hinesburg, VT, 05461 American 3
```

```
> reservations of "Henrys Diner"
no
> address of "El Cortijo"
189 Bank Street Burlington, VT 05401
> address of "Single Pebble"
no information
> quit
```

The important thing is that your query language will depend on the data that you've chosen. You can hard-code the expected tokens in your query (in my example, the words `ratings` and `cost` and `restaurant`, etc.). Your parsing code can be very picky: it will expect users to know your query language. It should give them useful feedback when they enter an invalid query. It should also have a little bit of help text to show users the structure of valid queries.

Note on my example: my names are case sensitive. I removed punctuation (e.g., apostrophes) from my text strings.

4.3 Specifications

Here are additional specifications. **Read these carefully.**

4.3.1 Command-based

Every interaction in the program must be in the form of a command in your query language.

4.3.2 Help command

One of the commands must be `help`, which will print out help text about the commands.

4.3.3 Multi-word data

Your system should handle data elements that consist of more than one word.

In my example, such data includes state names such as El Cortijo and Henry's Diner; and so the query language should allow the use of quotation marks to delimit these multi-word elements. But it's OK not to handle quotes or punctuation inside of tokens (again, as my example shows, I removed quotes and punctuation from restaurant names).

4.3.4 Query breadth

Your system must enable a user to obtain data from at least three fields of your top-level collection.

4.3.5 Compound (AND) queries

Your program must allow a queries that specify values in more than one field: documents for which $field_1 = value_1$ and $field_2 = value_2$. The `FieldFilter()` interface will be useful here.

4.3.6 Optional fields

Your program should gracefully and correctly handle the fields that are optional.

4.3.7 Robustness

Your program must be robust: if I put in a command that the program doesn't understand (*i.e.*, a command that doesn't follow the expected legal syntax), then the program should tell me (and not crash!).

4.3.8 Data source

The data must come from a JSON file.

4.4 Security

The simplest way to implement security authorization is by generating a project-specific private key, which takes the form of a JSON file.

See the section "Initialize the SDK in non-Google environments" in this reference. If you do this, give your key file a name specific to your project team.

You can certainly investigate alternative authorization mechanisms, but whatever you choose should enable us (the TAs and me) to authenticate successfully.

5 Designing and Developing Your System

Start early. The technology and the interfaces will be new to most of you. Parts of this system are complex. If you wait until the last minute, you'll almost certainly run out of time to complete the project.

5.1 Components

Design your system in way that encapsulates the different components of the system.

- the data: select your data and clean it; or create your own data; and put it into a JSON file
- your datastore structure: building off of the structure of your data, design your document and collection structure
- the parser: design your parsing system (you can even prototype it) to be be independent of the datastore operations
- the datastore operations: create wrapper functions for the various operations you'll need to perform (load data, query)
- an interface between the parser and the query engine

This design can then lead to the actual code that you develop. Keep the parsing separate from the datastore queries.

5.2 File structure

You should have at least three separate Python files: one for Firebase connection and authentication; one for the admin program; and one for the query program. The latter two should use functions that you put in the first of these files.

Create a class to represent each of your entities. You can see an example of this here.

You can write an ad-hoc parser for the queries or use something more sophisticated, such as PyParsing; or write your own recursive-descent parser (see References).

If you streamline the interface between the parser and the query engine, then your team can do development simultaneously on both components.

5.3 Indexing

When you start doing queries, you might get an error such as this:

```
The query requires an index. That index is currently building
and cannot be used yet. See its status here: https://console.firebase.google.com/v1/r/project/restaurants-23451
    f/firestore/indexes?create\_composite=
    CmBwcm9qZWN0cy9yZXN0YXVyYW56cy01MDYxZi9kAXRhYmFzZXMvKGRlZmF1bHQpL2NvbGxlY3Rpb25
    Hcm91cHMvUmVzdGF1cmFudHMvaW5kZXhlcy9DSUNBZOppbTE0QUsQARoICgRjaXR5EAEaCAoEY29zdBABGgwKCF9fbmFtZV9fEAE
```

This means that an index must be created for the collection you are querying. Just follow the link that's provided, and the necessary index or indices should be created automatically.

6 References

Here are some references that I have found useful:

- https://firebase.google.com/docs/firestore
- https://google-auth.readthedocs.io/en/master/user-guide.html
- https://firebase.google.com/docs/admin/setup#python
- https://firebase.google.com/docs/firestore/quickstart#python
- https://www.booleanworld.com/building-recursive-descent-parsers-definitive-guide/

7 Software Development Process

I recommend that each team follow a simplified Kanban process. You can use Trello (an online system for managing collaborative development) or you can track your status more informally, such as on a Google sheet.

I will meet with each team (schedule to be posted on Brightspace) for ten minutes or so on Wednesday, Sept. 4^{th}, and Friday, Sept. 6^{th}.

8 Deliverables

There will be four deliverables for the project.

8.1 Description of data

By Saturday, Sept. 7^{th}, at 11:59 pm: a description of the data you'll use, including the the name of the top-level collection; and a description of the entities in the top-level collection.

8.2 Query spec and interface spec

By Saturday, Sept. 14^{th}, at 11:59 pm: a short document describing your query language. List the keywords that users of your system will use, and provide several example queries. Also, describe the functional interface you'll use between the parser component and the datastore component. For full credit on this deliverable, show the specific function(s) and describe the parameters and return value of the function(s).

8.3 Individual report

By Wednesday, Sept. 25th, 11:59 pm: a short individual report, which each student will submit. Answer the following questions in your report. Provide some detail—more than just a one-sentence answer for each question:

1. Describe your team's project management: how did you keep track of tasks and status?
2. What learning did you have to do for the project, and how did you do that learning?
3. What worked well in this team project?
4. What did not work well in this team project?
5. What should your team do differently next time?
6. List two things that you personally should do differently on the next project.
7. Do you think your team did better or worse than the average CS3050 team on this project, and why?

Also, provide an assessment in the form of a 5-4-3-2-1 rating of the effectiveness and contributions of yourself and of your team members. 5 is the best; 1 is the weakest. These are ratings—not rankings—and they will be kept confidential (I'm the only one who will see them).

I've put an example of an excellent individual report from a past semester in the class gitlab repo.

8.4 Completed product

By Wednesday, Sept. 25th, 11:59 pm: submit a link to your gitlab repo. In addition, submit your credentials JSON file (rather than putting it into your repo). Be sure the JSON file containing your data (for the initial upload) is in your repo.

9 Evaluation

The warm-up project is worth 10% of the course grade. Half of this will be based on team performance, and half based on individual performance.

9.1 Team performance

We will be evaluating the software itself:

- the quality of the design: how effectively you've separated and encapsulated the parsing and the querying
- the quality of the source code: readability, consistency, comments
- the quality and robustness of program: can we break it?
- the correctness of the program: does it do everything I've specified?

I will also evaluate your process.

9.2 Individual performance

For this, I will evaluate your report and use the feedback from your team members.

10 Oral Presentation

Each team will give a 10-15 minute oral presentation, describing their system. In the first part, the team should introduce their product, and what it can do. Then, the team will demonstrate their system. This demonstration will count toward their group grade for this project. In the final part of the presentation, the team will summarize their analysis of the development process:

- what worked well in the development process
- what did not work as well in the development process
- how the team would structure development differently next time
- how the team communicated

The team should present their system on a team member's laptop. Spend very little time describing your actual code—I'm really interested in the process of how you created your system.

These presentations will be on the evening of Thursday, Sept. 26^{th}, from 6:00 to 9:00 pm, in Votey 209.

There will be no class on Friday, Sept. 27^{th}.