Alan T. Grubb
CS496-400
Assignment 3 Part 1

Data to be Modeled:

The data that I will be modeling will consist of two categories of data each with many properties. The two categories of data will be Locations and Users. These two categories of data and their inherent properties will be utilized in building a functional web application for the final project of this course. The application logic will be built upon/around the description of each of these categories and their respective properties.

Locations

Locations will store all relevant data to describe and display a location object within the application. To make this more understandable, first the context of Location objects will be described. Location objects will serve as a places where Users (another category of data) may congregate to engage in pick up sports games. In order that locations may meet user's use cases and/or needs, it is important for locations to contain relevant kinds of information. Here is the data that will be modeled for the category of Locations.

1. Name
    a. Required.
    b. Unique - this must be implemented via application logic.
    c. Accurate with respect to real-world locations.
    d. Only one per location.
2. Image(s)
    a. Optional.
    b. May be many per location.
3. Rating (5 Star)
    a. Optional.
    b. Simplistic - not broken down further into ratings for different facets of the location.
4. Description
    a. Optional.
    b. Some limit to the number of characters allowed.
5. Sports/Games Available
    a. Required.
    b. Accurate with respect to the sports/games actually able to played at the location.
    c. Selected from a prepopulated list.
6. Comments
    a. Optional.
    b. Contain information about User that made the comment.
    c. May be necessary for a 3rd category of data to be implemented - Comments.

7. Spatial Location (GPS data)
    a. Required.
    b. Should be accurate although users will input this data.
    c. Potentially will utilize Google Maps in order to achieve?
8. Hours of Operation
    a. Optional.
    b. Accurate with respect to when it is legal to be active at the location.
    c. Maintained by users.

The above list serves as a base by which Locations will be modeled/implemented. This list may not be exhaustive, as the development process may reveal additional properties that are necessary to describe/model Location objects. Also, as mentioned in the lectures, use cases are dynamic and change. In accordance, the application logic and the modeling of data within a database must change as well. It is unlikely that this will influence this project, as the scope is too small and the time frame too short. Use cases are not likely to change throughout the duration of the project's development but it is still important to consider and keep in mind as we are learning about real-world web application development and the implications therein.

Users

User data will not be quite as complex as Locations. The application does not depend *as much* on robustly modeled user data and the included properties reflect that. Users will essentially require just basic information to be stored - enough that users can interact with one another to perform likely use cases within the application. So what are some of these use cases that will inform the description of User properties? Obviously it is important that users have some kind of unique name/handle/tag by which they can be recognized. Additionally email or some type of contact information would be useful in order to facilitate some forms of communication between users. The description of User objects and the data they store *could* stop there and likely the application could still accomplish most if not all use cases desired by users. To make the project a little more interesting and add more substance to the application more properties will be included. The list of properties below is again not exhaustive and may change somewhat during the project's development process.

1. Name
    a. Required.
    b. Unique - implemented via application logic.
2. Contact Information (email)
    a. Required.
    b. Unique - implemented via application logic.
    c. Accurate or in the correct format.
3. Images
    a. Optional.

    b. Multiple allowed.
4. Sports/Games Interested In
    a. Optional.
    b. Selected from a prepopulated list.
5. Rating (5 Star)
    a. Optional (default null).
6. Comments
    a. Optional.
    b. Contain information about User that made the comment.
    c. May be necessary for a 3rd category of data to be implemented - Comments.

Plan to Model Data:

The plan to model the two categories of data described above is via using Google App Engine’s non-relational Datastore. This is fairly straight forward as far as implementation and Assignment 2 was the first step along the path to storing data within the datastore. As it stands, Locations have been implemented on a very basic level and some of the aforementioned properties are stored about these Location objects. Currently the functionality only exists such that users may use a form submission (via POST request) to store a Location with the following properties: Name, Active (boolean), Image (Only one), Star Rating and Description. The logic for creating these objects is coded with Python and utilizes the Google App Engine framework/libraries (ndb). The code for this will be modified such that users of the application can create Location objects that contain all of the properties described for Locations within this document. The code which implements the creation of Locations will be leveraged to allow for the creation of User objects.

The data stored for Locations and Users will be represented via JSON with key value pairs, which is easily human readable and understandable. This is how users will interact with data in the datastore when the API is created for next week’s assignment. An example JSON blob that would be returned for Location and User objects would look like this…

```
“User”: {
        “Name”:”Alan Grubb”
        “Contact”:”grubba@onid.oregonstate.edu”
        “Images”: {
                “Bytes”:””
        }
        “Sports”:”Soccer, Basketball, Tennis”
        “Rating”:”4”
        “Comments”: {
                “Comment 1”: {
                        “User”:”Dan B.”
                        “Timestamp”:”2015/2/11 9:35:12 -0700”
                        “Body”:”Friendly and pretty decent at tennis. Would rematch!”
                }
        }
    },
    “Location”: {
        “Name”:”City Park”
        “Images”: {
            “Bytes”:””
        }
        “Rating”:””
        “Description”:””
        “Sports”:””
        “Comments”: {
            “Comment 1”: {
                “User”:”Alan G”
                “Timestamp”:”2015/4/11 10:30:12 -0700”
                “Body”:”Great place for tennis!”
            }
            “Comment 2”: {
                “User”:”Whitney J”
                “Timestamp”:”2015/5/11 1:30:11 -0700”
                “Body”:”I liked this place. Clean and good for soccer!”
            }
        }
        “GPS”: {
            “Latitude”:”<somedata>”
            “Longitude”:”<somedata>”
        }
        “Hours”:”7am-10pm”
    }
```

From the JSON notation, something becomes abundantly clear - we’re going to need more objects. For both Users and Locations, Comments are going to be implemented as their own objects with three inherent properties (User, Timestamp and Body of the comment). For User objects, this is really the only additional object we need to model aside from images. For Location objects, however, there will be a need to utilize GPS objects that contain necessary information like latitude and longitude. Further research needs to be done in order to see what these objects for GPS data might look like. The plan is to utilize the Google Maps API to accomplish this task and model the GPS data. Images are pretty straight forward and will only contain the byte/binary data for the image itself. It is beyond the scope of this project to implement image objects that are any more robust than this. Again, more research needs to be done regarding how to return images as JSON objects but from a cursory look it is at least obvious that images need to be their own objects.

Additional Non-relational Database Research:

Amazon DynamoDB

DynamoDB is similar to Google App Engine's datastore in that they are both (obviously) non-relational databases with a schema-less structure. Also, both are hosted by huge companies with many (many!) nodes available to the end user/developer. Both of these are highly scalable technologies and that is a primary selling point to the customer for these cloud database solutions. Apart from many of the obvious similarities, these two technologies do differ in some important ways.

One of the primary and most significant ways these two database solutions differ from one another is in property indexes. As was talked about quite a bit in the lecture material for this week, indexes are useful and important in order to make time efficient queries of the database. Without indexing, the time-cost of queries increases substantially. GAE's datastore supports both 'single and multi property' indexes while AWS DynamoDB does not support them at all. For indexing within DynamoDB, the developer must implement it themselves via tables of their own creation, design and maintenance. For the inexperienced developer (like myself) this is distinguishes the two technologies from one another and makes the GAE datastore more desirable.

Another important difference between Datastore and DynamoDB is that the latter does not 'have' or support transactions *across* entities. Transactions were alluded to in the lecture material and can be described as when we perform work on a database in a single unit. This corresponds to the ACID principle of atomicity. Transactions either happen in full or not at all. GAE's datastore allows transactions within entity groups, which were talked about in the second week's lectures. Entity groups are, for simplicity sake, how GAE datastore implements consistency. Entities belong to a parent group, which can be referenced when building the application's logic in order to ensure consistency. In summation, this is another important factor when considering AWS DynamoDB against the GAE Datastore. With GAE, the developer can utilize groups and perform transactions on multiple entities. With DynamoDB, transactions may only be conducted on individual entities/entries.

All-in-all, there would not be a significant amount of change in the application logic or the modeling of data between utilizing these two technologies. At this point the project and described data are too infant in scope to tease out how the differences between the two would impact implementation/design to a large degree. With that said, however, I surmise that as the project grows that Google App Engine's Datastore will become more desirable because of it's ease of use regarding indexing. Indexing is something I can already understand as being an important consideration as it heavily influences time-efficiency of queries. The fact that GAE Datastore has native support for indexing means that I can essentially 'write it off' and not have to worry about that aspect of design. So, in that sense, GAE Datastore influences my design by the omission of considering how to implement indexing on my own within the database.