**Requirements:**

**1.Cache**: Rather than immediately calling your remote service, first look in your local database to see if you have already retrieved the relevant information. If you find the information in your database, display it from there. If it isn't in the database, make your API call, store the results and then display the data as before. An example would be getting Spotify artist information, which typically requires a call to Spotify to get the Spotify artist ID, then another call to get the desired information. Artist IDs can be cached, since they should be constant.□

Don't worry about cache invalidation, which would add significant complexity. If you already have a data store set up, it probably makes sense to use that for caching, but teams are welcome to explore components such as Redis, an in-memory key/value store.□

*We are using MongoDB as a cache. From our research, we inferred that Redis is more flexible and data is often stored in specialised data structures. However, MongoDB is simpler to use and since we are dealing with similar type of queries in our web application, MongoDB is easier to implement and manage because the structure is consistent across the data. Hence MongoDB cache directly is used. This is how it works:*

1. *Client/User enters search query*
2. *Search will go find result of query in the MongoDB cache*
3. *If result to query is found, response is retrieved*
4. *Retrieved response ios then returned to client*
5. *If response is not found, then it would call the Yelp and other APIs implemented and then load this result instance into the search response collection in MongoDB.*

*These concepts are illustrated in Parts 2 and 3 below.*

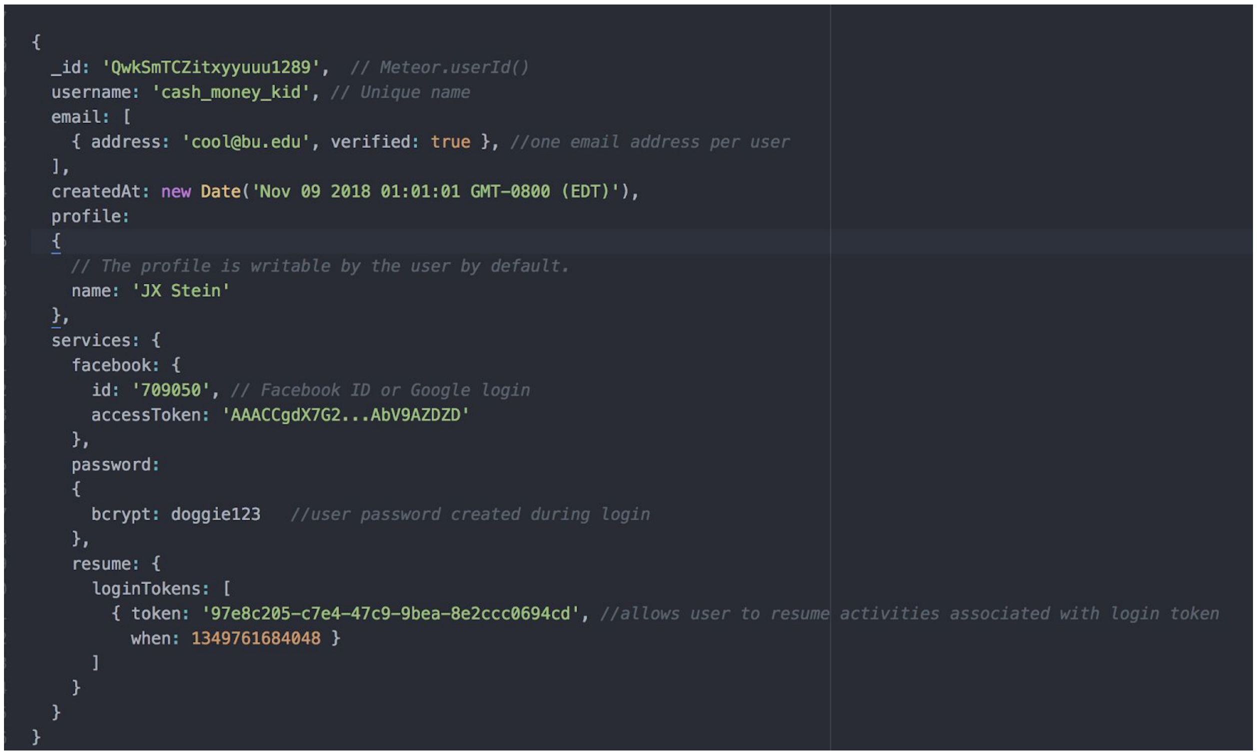
**2.Data documentation**: Add to your git documentation folder a document with your current and planned data models. If you are using a document-based data store such as MongoDB/ mongoose, your description can be a commented JSON schema describing the format and expected value of each field, which fields are required, and a note on which fields you will use for keys for each schema.□

If instead you are using a relational database (mySQL, Postgres etc), you should include an entity relationship model (ERD) that shows how each table is related to the others in your database, including formats, required fields, and keys.□

Don't forget to include information about your cache; it can be how you are planning to implement if it isn't in place yet.□

We can explain our data schema using 2 parts :

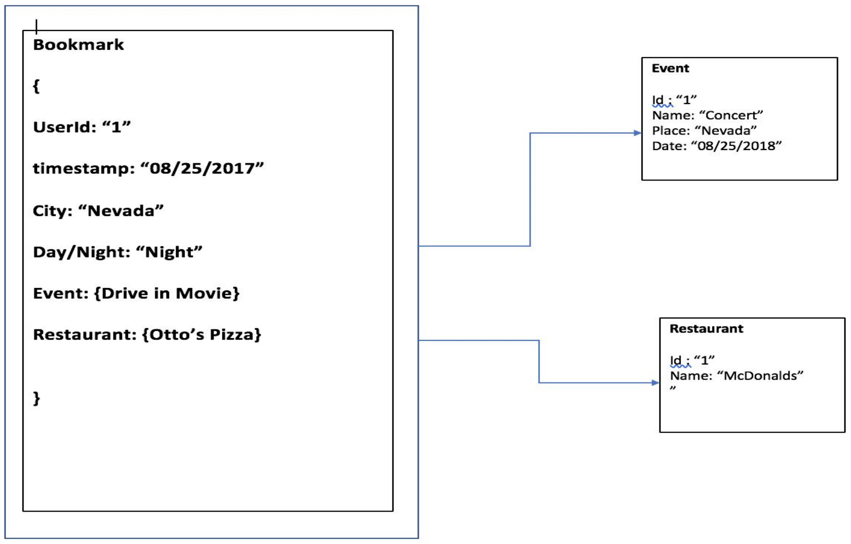
1. <u>The user authentication block:</u> Here, the user is able to create a profile by supplying a login name,email address,password and a facebook or google authenticated login.



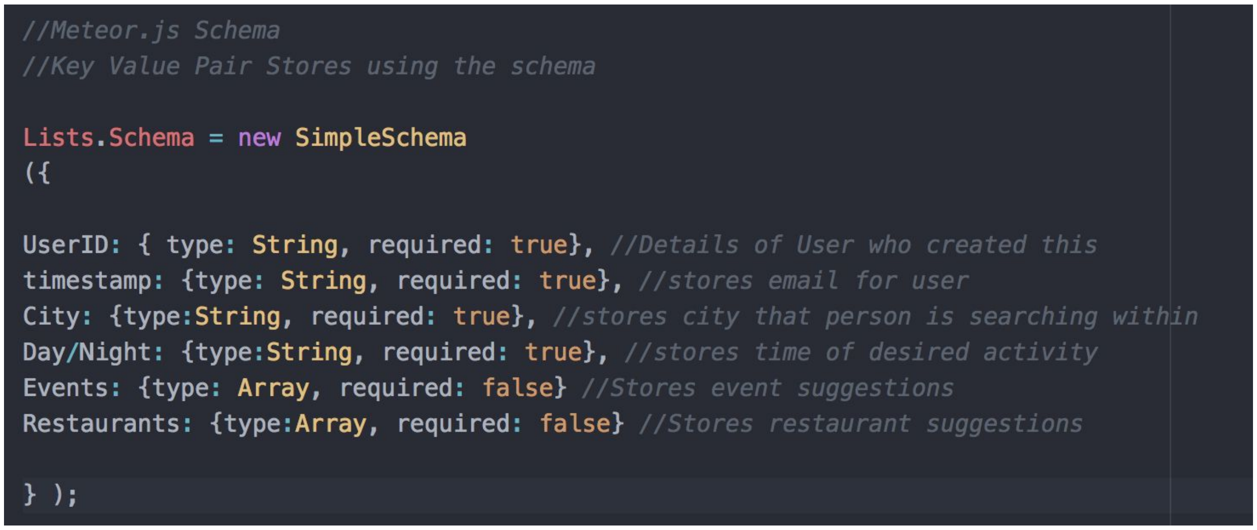
```
{
  _id: 'QwkSmTCZitxyyuuu1289',  // Meteor.userId()
  username: 'cash_money_kid', // Unique name
  email: [
    { address: 'cool@bu.edu', verified: true }, //one email address per user
  ],
  createdAt: new Date('Nov 09 2018 01:01:01 GMT-0800 (EDT)'),
  profile:
  {
    // The profile is writable by the user by default.
    name: 'JX Stein'
  },
  services: {
    facebook: {
      id: '709050', // Facebook ID or Google login
      accessToken: 'AAACCgdX7G2...AbV9AZDZD'
    },
    password:
    {
      bcrypt: doggie123   //user password created during login
    },
    resume: {
      loginTokens: [
        { token: '97e8c205-c7e4-47c9-9bea-8e2ccc0694cd', //allows user to resume activities associated with login token
          when: 1349761684048 }
      ]
    }
  }
}
```

<u>Figure 1: User Authentication Block Meteor.js</u>

2. <u>The bookmarked results output block:</u> After the user puts in the search query, there is an event result that is generated and a restaurant result as well which is later concatenated to shoe the user the best combination of those choices according to their search query. The boxed diagram view is shown below along with a JSON Schema below:



<u>Figure 2: Bookmart result output block diagram</u>

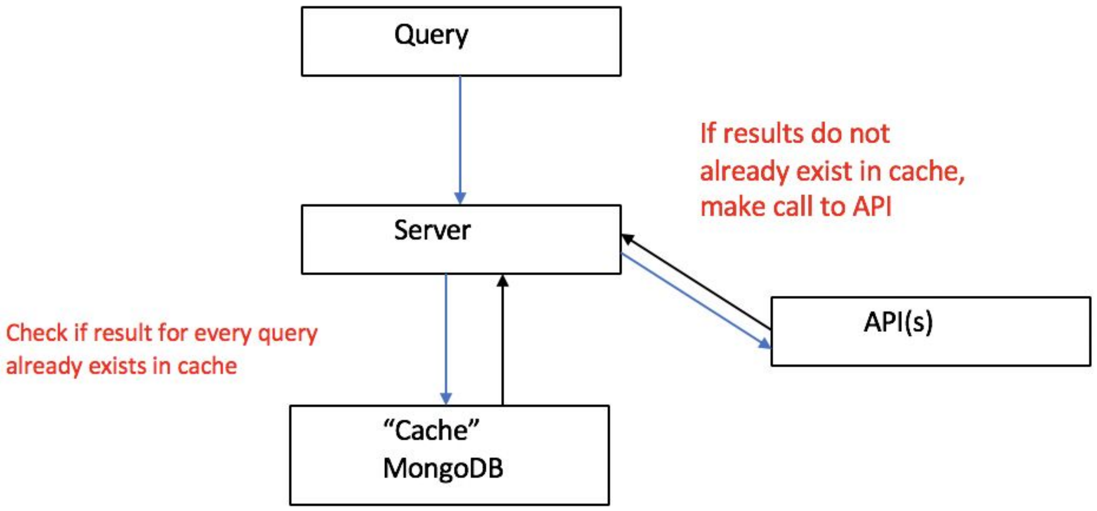


```
//Meteor.js Schema
//Key Value Pair Stores using the schema

Lists.Schema = new SimpleSchema
({

UserID: { type: String, required: true}, //Details of User who created this
timestamp: {type: String, required: true}, //stores email for user
City: {type:String, required: true}, //stores city that person is searching within
Day/Night: {type:String, required: true}, //stores time of desired activity
Events: {type: Array, required: false} //Stores event suggestions
Restaurants: {type:Array, required: false} //Stores restaurant suggestions

} );
```

Figure 3: JSON schema representation for key-value types

**3.Sequence diagram**: In addition to the diagrams from (2), create a sequence diagram that shows how an API call will be made using your cache, starting with the user entering a search term and ending with display of search results. It should include the cache step from this exercise. I think that StarUML is a good tool for this kind of diagram, but feel free to use whatever tool you are most comfortable with.

The  sequence diagram showing how an API call will be made using the cache is shown below:


Query
Server
If results do not already exist in cache, make call to API
API(s)
Check if result for every query already exists in cache
"Cache" MongoDB

**MongoDB Collections – We are using MongoDB as Cache.**

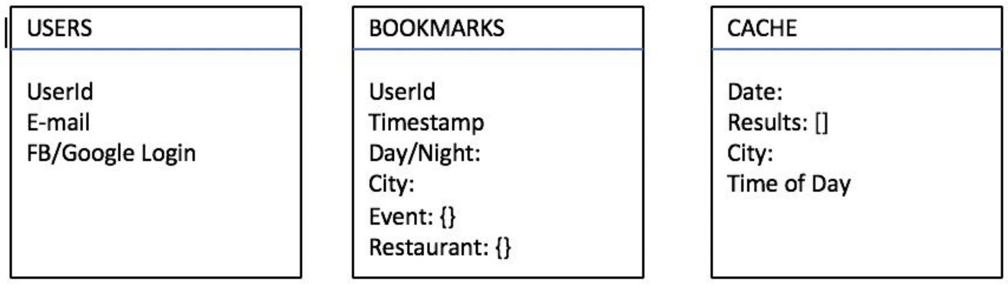


Figure 4: Cache Sequence of Events

To put the above diagram in perspective, it useful to study it with the user stories that have been explored in the previous assignment. In the first part of our user story, our user first searches for a planned experience using a "Query" and the query goes to the server. Then, to allow for the user to "browse search results and filter through them", the server communicates with the cache first to check if the resiult for exists in the cache. However, if results do not exist, it calls the API to return the result. With the return of the customised results, we go to the last part of our user story where the "user has found a planned experience and saves this experience to their profile". Another integral part of our user story if log in/sign up process where the user "logs in to their social media account" and created a profile so that our application can save their preferences and history using our choice of database MongoDB. The overall schematic communication diagram between the cache and the MongoDB Database is illustrated below.

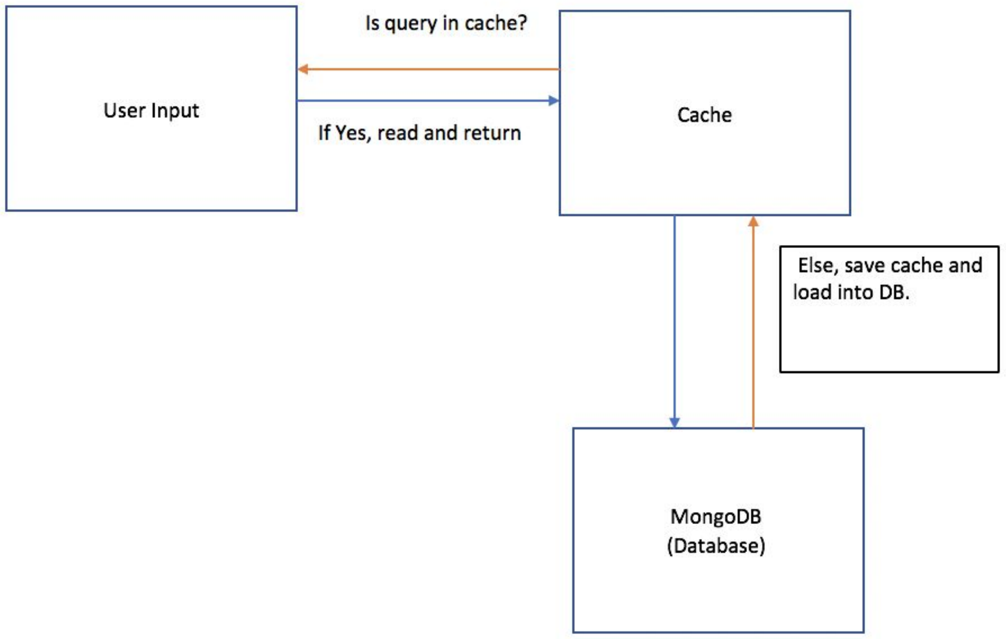


Figure 5: Cache-Database communication and sequence of events