Minhee Lee

Nishant Junanker

Alexander Goldman

Alexander Kornhauser

CIS 450/550 Final Report

1. INTRODUCTION:

The goal of this project was to synchronize our understanding of Relational Database planning, creating, and usage with a tangible website. We used the Amazon Web Services (AWS) and their Relational Database and EC2 tools as the backend of this website. With these tools we were able to implement our own schema in a way we considered to be most logical given the project specifications and description. This schema differed from the TA’s schema that was released in various ways, and one of the challenges was importing and exporting this data in a reasonable way.

To build the actual website we used Node JS and its various modules, most notably Node Oracle. Node Oracle allowed us to connect the JavaScript code to the database and thereby run the SQL queries. There were some issues with this, and getting this setup was one of the most difficult tasks, which will be highlighted later.

All in all, we were able to implement all of the features and we accomplished our goals of understanding the various aspects of utilizing a backend in creating the Tripster website.

2. DESIGN:

The design, without doubt was the key in creating the success of this project. There were various components of design that went into the website, and they can be enumerated separately.

The communication between the server and client side was accomplished by using the Node module Oracle. This module allows the calling of a connect function, which takes in a string as an SQL statement and returns an object containing either the return of the query, or it confirms that a field was inserted into the database.

Many other modules were important as well. The sessions module allowed us to have a single user (first set in the login.js or signup.js file) to be easily passed into the other routes, and thus keep track of the same user throughout the website.

The bcrypt module was also very important in maintaining a password hash instead of an actual password. We would hash the password with bcrypt and then add that password to the database.

Express and Stylus were also useful. This app is of course an Express app, and the Stylus Jade template proved easier to read and edit than the bulkier Hogan html.

The routes design is pretty self-explanatory. We basically would navigate from the app.js (modeling what was done in the sample actor file), to index.js, which is basically the home page. Index would connect to both signup.js and login.js, which then navigate to newsfeed.js, which is the main page. Users.js, like.js, and newsfeed\_trips.js work similarly.

We can also highlight how we designed solutions for certain problems:

1. Newsfeed ordering and Prioritizing:

For newsfeed ordering we used an algorithm that recorded how frequently a user tends to like another user’s post. The more likes, the higher priority, and there is also a metric for how recent something is that raises priority.

1. NoSQL usage:

We used Mongo and Mongoose, and used Mongoose to describe the schema of the images we will be storing. For each image there is an associated U\_ID detailing who posted the image, the src, and the buffered data.

1. Searching:

For searching we search for users/photos but user\_id of the user/photo up-loader. For locations we search by location by the location name, which is a primary key.

1. Implementation of Caching:

For each image, we find that the search yields multiple images, and we call a helper function called add\_photo\_to\_cache, which builds the item to add, and calls save.

1. Add photos/albums to trips with privacy settings:

Each trips contains with in itself a photo\_in\_trip\_id, which also connects to a photoID, allowing the private storage of photos in trips. This same logic is used in albums, where there is album\_share\_trip, which binds to album and trips.

3. TECHNICAL ISSUES:

The biggest problem we had was getting Node Oracle to install. Half of our group members accomplished this by using the Linux Image which had Node Oracle installed. However half of the groups tried to install it and found that it was very poorly documented. The chief documentation was a github account that clearly did not test the listed instructions on many platforms. Installing Node Oracle so that it worked correctly took longer than any other single task.

Another issue that we encountered was importing the data. The way we designed our database had a view differences with the data format. One issue was that we used different types for various fields, such as U\_ID, the primary user id key. We found it difficult to alter the data so we decided to alter the field in the database. This proved to be a tedious task in the formation of the database.

4. PERFORMANCE:

The relevant latencies for queries are listed below:

Loading the Newsfeed: 745.202 ms (highly variable)

Trips: 774.254 ms

Users: 602.691 ms

Newsfeed Trips: 79.058 ms

Update user: 208.291 ms

Login: 129.809 ms

Signup 134.018 ms

These results are all fairly similar to one another. However we saw some variability in the latencies, especially in regard to Newsfeed. Sometimes the newsfeed latencies surpassed a second, even 3 seconds (3224+ ms recorded). Thus the bottleneck is clearly in the newsfeed, as it makes various calls and loads a significant amount, relative to some of the other queries.

5. FUTURE EXTENSIONS/EXTRA CREDIT:

There could be various additions added on the to website. One such is incorporating Facebook, which would have worked very well with the sessions module that we used to keep track of the current user logged in. We did not, however, complete any of the extra credit assignments for the project.