Lisa Percival Final Project

Email: percival@onid.oregonstate.edu

Link to demo video:
http://web.engr.oregonstate.edu/~percival/CS496/CS496_FinalProject.mp4

Overview

My final project builds upon work done previously for assignments and is an updated version of an Android application that uses academic reviews to rank schools. The biggest changes from the previous version involved adding user accounts and limiting some data and actions to only be visible to a logged in user. Anyone can, without logging in, view a list of ranked schools and add a school. After logging in, a user can create reviews, view the reviews they wrote (and not any someone else wrote), and edit and delete those reviews.

The mobile feature is use of the SQLite instance on the Android device for temporarily storing some data. A user can still, on the Add Review page, save a draft of the information they've put in the form and later load it back in before submitting the review. The draft information is stored in a SQLite table, and is now specific to the logged in user. If another user saved a draft, you can't load it. I also expanded use of the SQLite instance so it serves as a partial cache of the list of schools. In the previous version, the dropdown list of schools on the Add Review page was populated by a school list sent as an extra on the Intent for that Activity. However, that was less than ideal and got complicated when adding the Edit Review page that needs the same dropdown and different ways of navigating between pages. Therefore, the Ranked Schools page now caches the list of school names and IDs that it gets from the API in a SQLite table that is accessed by the dropdown lists on the Add Review and Edit Review pages. That same table is also used on the My Reviews and View Review pages to quickly replace the school ID stored in a review with its name, which is much more user-friendly.

Another feature I would like to mention is the fact that the action bar dropdown of menu items (the … in the top right) changes based on whether or not someone is logged in. It swaps between Login and Logout based on

what's appropriate, and only displays the options for My Reviews and Add Review if there is a logged in user.

Any errors encountered are displayed with Android's toast messages (little popups near the bottom of the screen). Most error messages for bad attempts to add or edit data come as responses from the API but the user would never know. Toast messages are also included for informational purposes.

API

The API used is yet another iteration of the one that I originally created back in Assignment 3 and adapted for Assignment 4. In order to implement my user account system, I had to do some pretty significant restructuring of the API, including the URIs and non-relational database. The cloud back-end is still implemented in Google App Engine with the NDB Datastore, but I changed the entities and how they are related.

The first adjustment was to add a User entity, which represents a user's account and has two string properties: a username and password. My other two entities are School and Review. A School still has strings for name, city, and state, as before. It also has two new properties, which are a calculated average ranking and a list of rankings from associated Reviews that can be used to update the average. Reviews contain the properties they had previously, namely a string title, string description, date, and integer ranking, as well as a new KeyProperty that relates it to the School it is about.

The motivation for these changes began when I decided to implement my own account system, have Reviews belong to a particular user, and display a page with all the reviews for a particular user. In the previous version, Reviews had the School they applied to as their parent, so I had an entity group of Reviews for each School. However, sticking with that approach would have meant that the My Reviews page would have had to iterate over all of those entity groups and pull out the Reviews somehow associated with a User. It made a lot more sense to group Reviews into entity groups based on the User that authored them, and have them relate to Schools (in a separate group) with a KeyProperty instead. Therefore, the final structure is to have one entity group of Schools that simply has a parent key of "schools", one entity group of Users with the parent key "users", and then

an entity group of Reviews for each of those Users that have the User's ID as their parent key. One advantage of this solution is that there will be guaranteed consistency on the My Reviews page. This approach also makes for more reasonable/ RESTful URIs and makes it less likely there will be an issue with going over the one write per second per entity group limit.

Of course, Reviews also still need to be related to Schools, and the list of Ranked Schools needs to show the School's average ranking based on those Reviews. I could have tried to calculate that on the fly when the list of Schools was requested, like the previous version did, but it would have required iterating over all the Review entity groups and somehow tying them back to each School. Since the focus of non-relational databases is speed, especially of reads, and it's OK to duplicate some information, I decided to instead store the average ranking of each School as a property. This means that the School's average ranking must be updated whenever a Review related to it is added, edited, or deleted. It does add some computation to those operations, but it solves the issue of how to backtrack from a School to its Reviews and also means there's no concern about consistency in the Ranked Schools page. In order to effectively update a School's average ranking based on Review actions, I also had to store a list of all its rankings so the average can be correctly re-calculated when necessary, which means that list also has to be updated when a Review is added, edited, or deleted for that School.

URIs

- http://stately-list-96223.appspot.com/schools
    - GET: obtains the list of ranked schools, as displayed on the Ranked Schools page
    - POST: adds a new school, used by the Add School page
        - Can have 3 string parameters: name, city, state
        - Name is required
- http://stately-list-96223.appspot.com/users
    - GET: obtains the list of users, used in the Android app for login to check whether given username and password are in valid list
    - POST: adds a new user, used by the Sign Up page
        - Must have 2 string parameters: username and password (called pwd)
        - Checks if username is already taken
- http://stately-list-96223.appspot.com/users/[user_ID]/reviews

    - GET: obtains the list of reviews that belong to the user with the given ID (who's logged in), as displayed on the My Reviews page
    - POST: adds a new review, used by the Add Review page
        - Must have string title, string description, date, string containing school's ID (called school), and integer ranking
        - Author is automatically obtained in Android app based on who is logged in
- http://stately-list-96223.appspot.com/users/[user_ID]/reviews/[review_ID]
    - GET: obtains the details for the single review given by the ID, as displayed on the View Review page
    - PUT: updates the given review with the new data, used by the Edit Review page
        - Must have a string title, string description, date, string containing school's ID (called school), and integer ranking
        - Must also send the previously-set ranking and a string with the previous school's ID (called oldRanking & oldSchool), because they're used to update average rankings
    - DELETE: deletes the review with the given ID, used by the Delete button on the View Review page

REST

This API is not perfectly RESTful, but reasonably so. The biggest thing is that it has a resource-based URI structure, which is based on plural nouns and reflects the relationships between resources. Operations are performed on the resources, identified by their Datastore keys, using the HTTP verbs. It is also stateless and satisfies the constraint of separation between server and client. The API is purely a server and concerned only with data, whereas the Android app as the client is all about the user interface.

Account System

I implemented my own user account system. Accounts are stored as User entities in the Datastore of the cloud API. This allows different users to be able to view, add, update, and delete data that is specific to their account. This data is reviews, which are associated with accounts as described in the API section. Users can also create a new account.

To handle user login in the Android app, I took advantage of a feature called Shared Preferences, which allows for temporary storage of key-value pair data. I had a lot of assistance from http://www.androidhive.info/2012/08/android-session-management-using-shared-preferences/. One advantage of using Shared Preferences is that they are private to the application but persist even when the user closes the application. The persistence is a tradeoff when it comes to security, but would be nice for usability. By storing, retrieving, and deleting 2 key-value pairs in Shared Preferences, I was able to create sessions that track whether someone is logged in and, if so, who. One piece of data is a Boolean that indicates whether a user is logged in and the other is a string that contains the ID of the logged in user when applicable. I created a session class that can log in a user by setting that data in a Shared Preferences file and log out a user by clearing that data. It can also determine whether someone is logged in, get the ID of the logged in user, and redirect to a different page if no one is logged in. Then I was able to make use of the functions from that class in all of my Activities to handle account management.

The main Ranked Schools page can be viewed without being logged in, but checks whether anyone is logged in in order to populate the action bar menu items based on what actions can be taken by a non-logged-in user. If you're not logged in, all you can do is view the schools, add a school, or log in. The Add School page can also be used without being logged in, but has the same check and the only option shown in the action bar will be to log in.

If someone could somehow access the My Reviews, View Review, Edit Review, or Add Review pages without being logged in, in spite of the fact that they do not show up in any navigation path, they include a check for a logged in user that would cause redirection back to the Ranked Schools page.

After selecting the Login option, you can either provide an existing username and password to log in, or click the Sign Up button to add a new account. If you attempt to log in, the app will get the list of current usernames and passwords from the API and validate what you provided against it. If there is no match, you get a toast saying your credentials are invalid. If there is a match, the app logs you in using the session class and redirects to the Ranked Schools page.

On the Sign Up page, both username and password are required. The information you enter is sent to the API with a POST request where it validates they were both provided and that the username does not already exist in the Datastore. If there is an issue, you are notified with a toast. If the API is able to create the new User entity successfully, the app logs you in using the session class and redirects to the Ranked Schools page.

When a logged in user views the Ranked Schools page, the action bar menu now shows options for My Reviews, Add Review, Add School, and Logout.

The My Reviews page uses the session class to determine the ID of the logged in user, and inserts that into the URI for the GET request to the API to obtain all the Reviews associated with that User, which are then displayed. Clicking on one of the reviews shown takes you to the View Review page, which also uses the session class to determine the ID of the logged in user and adds it, as well as the ID of the review selected, into the URI for the GET request to the API to obtain the data for that particular review. If you press the Delete button on the View Review page, it again uses the logged in user and review ID in the URI for the DELETE request to the API. If you choose the Edit button, it takes you to the Edit Review page, which also uses the session class to get the ID of the current user and inserts that and the review ID into the URI for the PUT request to the API.

Similarly, the Add Review page uses the session class to obtain the ID of the logged in user, and includes it in the URI for the POST request to the API. This causes the new review to be added in the entity group that has that User's ID as the parent key.

When a user chooses to log out, the app simply makes a call to the session class function that deletes the login data from the Shared Preferences file and then redirects back to the Ranked Schools page.