Zachary DeVita

CS 496

Final Project

Video: <https://youtu.be/ebCa00xs--I>

For my final project I used my Space Shooter Unity game as a front end and I used a Node.js server-side architecture for my backend. As far as the database goes, I used non-relational DynamoDB database to hold my data. The mobile platform for the front end is Android. The project went pretty well, I don’t know if I’ve ever learned so much in a two week period. Using Unity to create my front-end UI menus proved to be far more difficult than I had anticipated. Much of the work was not able to be cut & pasted as with programming so when I had created some error between the server and Unity, and it took me hours to solve, it would commonly take me hours to solve for each component which exhibited the same behavior.

The mobile feature I implemented is utilizing the mobile device’s accelerometer. During gameplay the spaceship is completely controlled using the accelerometer. The screen is pressed to fire. I have a few one-to-many relationships incorporated in my server-side programming and database tables. For instance, I have a user table and a userscores table, and I am able to access many scores associated with each user account. I also utilize batch get and batch write functions in my node.js script. These allow me to update and retrieve multiple items from multiple tables with a single query. My front end has been modified to work on an Android app. The video is shown on screen capture software using an Android emulator for my computer but the game build is an .apk file. The resolution used in my video is at 540x960, but my app will auto-adjust the resolution to fit whatever device it is running on.

My front-end consists of a login page, a registration page, an account page, a separate page for editing each item of personal data, and, of course, the game. The login page has two inputs; one is for the user’s email and the other is for their password. There are two buttons as well. One takes the user to the registration page, and the other will attempt a login with the user’s credentials. There are error messages for incorrect input which in Unity is not a simple task to implement. The registration page has input fields for the user’s personal information, and a button to click when they’re done. The profile, or account, page shows the user’s personal information, as well as, their top ten high scores in the game. From here they are also able to edit their username, password, and/or phone number. They also have the option to delete their password with a button click. The user can also delete their account from here or logout, or play the game with a button click.

My back-end has the URIs which make the front-end stuff possible. I use one batchGetItem command which retrieves data from both of my tables with one query. This seemed to be a requirement previously so I made sure to get it this time. I also have written an algorithm, which uses the Passport library, to create a password encryption and open up a session-state for the user so they maintain their authentication until they logout. I have functions which serialize and de-serialize data being transferred. I wasn’t able to find any way to return data from the table in sorted order, so I had to make functions to brute-force sort the user’s high scores.

As far as my restful API goes, I have implemented many URI calls in order to get my UI working correctly. It should be known that Unity only has support for GET and POST requests. I did not know this ahead of time, and I know it was OK’d by the instructor to use Unity. I was able to find a work-around for deleting the user’s phone number, but it is via a POST request and not a DELETE. I was able to set up the delete account URI using a get request.

GET

app.get('/get-account')

This URI requires the user’s email as a parameter and it grabs all user information from both the user table and the userscores table. The userscores data is then sent to an additional function to be de-serialized and sorted. This data is then sent in a single JSON object back to the front-end to populate the user’s account page.

app.get('/logout')

This URI requires no parameters. It simply closes the user’s session state, and, on the front-end it kicks the user out to the login page.

app.get('/remove-user')

This URI is my workaround DELETE URI. Because Unity does not have support for the DELETE functionality, I had to use a GET request to fulfill the same purpose.

POST

app.post('/login')

This URI requires the user’s email and password. This URI looks for a matching email in the database, then returns the encrypted password from the database, decrypts it, compares credentials, sets up the user’s session state, and gives the OK for the front-end to transfer them to their account.

app.post('/update-info')

This URI can be used for either changing a password or changing a username. It requires the 3 parameters, the datatype (the database attribute name), the user’s email, and the value that the attribute is being changed to.

app.post('/update-password')

This URI had to be separate from the previous because of the requirement to encrypt the password before it is stored in the database. It essentially does the same thing except it doesn’t require a datatype since we know it is a password.

app.post('/signup')

This URI takes the user input, checks to see if the email has already been taken, then encrypts their password, creates a table entry in userscores using the username, serializes the data and sends the data to the database. It requires all pieces of personal information which were entered in the UI to be sent as parameters.

app.post('/add-score')

This URI is used to store a new score into the userscores table for the specific user.

Just a note; my native code segments need the Unity engine to be ran. They will be included as C# files along with the server.js file.