Project 4 – Supeable Admin

In this project, you will return the "Supeable" scenario. In Project 2, you created the Supeable Mobile client. In this project, you will complete the code necessary for a Supeable Admin Client.

1. A working admin client will be demonstrated in class.
2. You will be provided with working client-side code for the admin interface. This can be found on the community site in a file named: supeableadmin.zip.
3. You should unzip this file, and upload to the public S3 bucket with the name supeableadmin_***lastname*** (where ***lastname*** is your last name!)
4. For the client-side code to work, you will need to replace endpoint01 with the URL to their own endpoint. You do this later. Just don't be alarmed if your client-side code doesn't work right away!
5. You will be provided with working AWS lambda code. This can be found on the community site in a file named: project4.zip.
6. You will use this file to set up a Lambda function named project4-***lastname*** (where ***lastname*** is your last name!)
7. You should set up a corresponding Web API gateway, and proxy gateway, as we have done in the past.
8. The database options in this file should be set to point to the same "blue" database you used in project 2.

CONTINUED

9. The features for the provided Lambda function are in the following table. Features marked with ** indicate features which you are expected to write.

Issue a GET against ./**auth** and provide a username and password. The API will respond with a JSON object containing all the user information except the password."
Issue a POST against ./**events** to create an event. You must provide the following keys: eventname, eventnotes (may be an empty string), createdby. The key createdby is the userid of the person creating the event. The API will return a JSON object representing the event that was just created.
Issue a GET against ./**events** to get a list of current events created by the user. You must provide the following keys: createdby, archived. The key archived will be a Y or an N. The API will return a JSON object containing all the events created by that user.
** Issue a PATCH against ./**archivestatus** to update the status code of the event. You must provide the following keys: eventcode , createdby, archived. The archived key will change to Y if it is archived or N if it is being unarchived ('not archived' is the same as 'current'.)
** Issue a GET against ./**attendance** to get a list of users that have attended the event. You must provide the following keys: eventcode and createdby. The key createdby is the userid of the person who created the event. The result will be a JSON object containing the eventcode, eventname, username, email, and the datetime the user event attended the event. These results are sorted by username, and also by attendance date. "

10. You should test their work with the Thunder Client. When that works, assign the URL of your endpoint to the variable endpoint01 found in app.js.
11. PLEASE NOTE – there is no client-side implementation to generate an attendance report. However, you are still expected to create this web service feature.