Lane Bryer

CS 496

Final Project – Cloud only

Third party service usage

The third party service used for account authorization/authentication is Google OAuth2.0 with the google+ API. You can utilize Postman for generating an OAuth2.0 access token. The API will take this token and send it to the Google+ API requesting information on the logged in user. It will then use the user's email address to verify ownership of datastore resources. Every API call will utilize this method to verify the user's email address matches the email associated with the owner of a given resource.

Setting up Postman

In order to make the test collection run, the grader will need to generate two different access tokens associated with two different emails. In order to do this, the grader select an auth type of "OAuth 2.0" under the Authorization tab of any given request. The grader should then click "Get new access token". All information will be prepopulated and the grader should simply then hit "request token". The grader will need to log in with a Google email account. The grader can then copy the access token string and paste it into the user1Token environment variable. The grader should then clear all cookies (click cookies and click the x on all cookies). After this, the grader should repeat the process with a different Google email account. If you do not clear all cookies, it will simply generate a new token for the email you used previously. You must clear the cookies in order to be prompted for a new email address. Again, copy the access token and this time paste it into the environment variable user2Token. The grader should then be able to run the test collection successfully. This process will be shown in the attached video in case this description is not sufficient.

Routes

GET /account:

This handler returns the user account associated with the email that the user is currently logged in on. There should only ever be 0 or 1 accounts returned as the user cannot have multiple accounts with the same email address. Since this is the case, there is no reason to provide a resource ID to find a specific account – it will always simply pull your own account.

POST /account:

This handler will create a new account associated with the current user. The body fields the user must provide are: firstName(string), lastName(string), and age(integer). An email field will be populated with the user's current email address. An array of todo items will also be generated (initialized to empty). If an account with this email already exists, an error message will be returned. If the user tries to create values in the array of todo items as part of the POST, an error message will be returned. If the user does not specify any of the three required fields (firstName, lastName, age), an error message will be returned.

PATCH /account/{id}:

This handler will update information for the user whose resource id matches the provided id. The user can provide any combination of the three editable fields (firstName, lastName, age) to be updated. If the user tries to specify a new id, email, or list of todos, the handler will return an error message. If the email address of the account id provided to the handler does not match the current user's email address, an error will be returned. This is simply to demonstrate that email verification is happening – realistically the user wouldn't need to provide an id as the API could simply update the account associated with the user (if there is one). If the id provided does not belong to any object, an error message will be returned. If the user does not provide any id, an error message will be returned.

DELETE /account/{id}:

This handler will delete the account object associated with the provided id. It will also delete all todo items associated with the account since todo items must belong to an account. Again, needing to provide an id here is just for demonstration purposes to show that you cannot delete an account that does not belong to you. If the owner of the account does not match the current user's email address, an error message will be returned. If the id provided does not belong to any object, an error message will be returned. If an id is not provided, an error message will be returned.

GET /todo:

This handler will return a list of all of the current user's todo items. If there are no todo items for the user, a message will be returned stating that you currently have no todo items.

GET /todo/{id}:

This handler will return a specific todo item that has an id provided by the user in the route. If the current user's email does not match the owner field of the todo item, an error message will be returned. If the id provided does not match any todo object, an error message will be returned.

POST /todo:

This handler allows the user to create a new todo item associated with their account. The user must specify "body"(string) and "dateToDo"(string). If the user tries to add "owner" or "dateCreated" an error is returned – these are read-only values. If the user does not have an account, an error is returned.

PATCH /todo/{id}:

This handler allows the user to modify an existing todo item. The user must provide an id for this handler. The user can specify any combination of "body" and "dateToDo" to update the todo. If the todo is not associated with the currently signed in user, an error message will be returned. If the user tries to specify "owner" or "dateCreated" fields to update, an error message will be returned. If the provided id cannot be found in the datastore, an error message will be returned. If no id is provided, an error message will be returned. If no errors are encountered, the specified todo will be updated.

DELETE /todo/{id}:

This handler allows the user to delete an existing todo item. The user must provide the id of the todo item. If the todo item does not belong to the currently logged in user, an error message will be returned. If the provided id cannot be found, an error message will be returned. If no id is provided, an error message will be returned. If there are no errors, the todo item will be deleted and the todo item's key will be removed from the associated user account's todos array of key properties.