Table of Contents

URL for App Demo Video

View or Download at:
http://web.engr.oregonstate.edu/~swansonb/BrandonSwansonCS491-FinalProject.mp4
I recomend wating on Youtube at 1.5x speed as the typing of user credentials slows the demo:
https://www.youtube.com/watch?v=OWdkdwBfLAo

Platform Used for Mobile Development And Server RESTful Backend

For this assignment I continued the development of the Youtube Snippet Playlist application that I developed in earlier assignments. The backend makes use of the Google App Engine Platform-as-a-Service, and uses their NDB set of classes to save and retrieve entities in a non-relational database.
There is both a mobile native (Android application) and an HTML rendering front end available for these entities. Conforming to RESTful principles the entities in the non relational database are representation agnostic, allowing the user to view the entities in a mobile application or browser.

API Summary

API root: http://swansonbfinalproject.appspot.com/
This app makes use of a modified version of the API I created for assignment 3 and used in assignment 4 also.. It is an API that allows for the creation of YouTube video snippets, a video from youtube with a user defined start time, end time, title, and notes. These snippets created by the user are also grouped into an ordered playlist created by the user. It should be noted that while YouTube provides a method for creating and sharing playlists of videos this API goes beyond that functionality by allowing the user to specify starting and ending times as well as

provide annotating notes. This API could be used for educational purposes, highlighting important 1-2 minute sections of a 2 hour lecture for example.
The API has 2 entity types that implement restful APIs (have entity urls and receive GET/POST/PUT/DELETE verbs at those urls). These RESTful entities are: Playlist, and Snippet. Featured below is a summary of their attributes and available API calls.

```
class Playlist(ndb.Model):
    isPublic = ndb.BooleanProperty(indexed=True, required=True, default=False)
    title = ndb.StringProperty(indexed=False, required=True)
    creator = ndb.StringProperty(indexed=False, required=True)
    snippetKeys = ndb.KeyProperty(kind=Snippet, repeated=True, indexed=False)
    date_added = ndb.DateTimeProperty(auto_now_add=True)
class Snippet(ndb.Model):
    title = ndb.StringProperty(indexed=False, required=True)
    notes = ndb.StringProperty(indexed=False, required=False)
    videoID = ndb.StringProperty(indexed=False, required=True)
    startTime = ndb.IntegerProperty(indexed=False, required=True)
    endTime = ndb.IntegerProperty(indexed=False, required=True)
```

URL	**Entity**	**Verb**	**Result**
/	Playlist(s)	GET	Returns HTML representation of all public playlists
/playlist.json	Playlist(s)	GET	Returns JSON representation of all public playlists, or user's playlist if authenticated with token
/playlist/<id>[.json|/]	Playlist	GET	JSON or HTML representation (depending on url suffix)
/		POST	Create new Playlist belonging to user, return new entity's URL Must be authenticated with token
/playlist/<id>/	Playlist	DELETE	Delete playlist and any associated Snippets Must be authenticated with token
/playlist/<id>/	Playlist/ Snippet	POST	Create new Snippet, associate with Playlist, return new entity's URL Must be authenticated with token
/snippet/<id>/	Snippet	PUT	Update Snippet information Must be authenticated with token
/snippet/<id>/	Snippet	DELETE	Delete Snippet and remove from associated Playlist Must be authenticated with token

The mobile front end interface provides the user with the ability to log in and make authenticated calls to GET/POST/PUT/DELETE entities which they created. And playlist entities which have been created as public can be retrieved and viewed (GET) as html or json. Snippets are created and associated to a playlist by posting the required fields to a Playlist url which returns that new Snippet entity's URL.

User Account Summary

For the creating and retrieving of user accounts I made use of a few classes within the WebApp2 library (auth, User Models). These modules provide some utility functions for creating a user with an associated password and retrieving that user entity by name and password or by token. But the user models are stored within my apps datastore and the api endpoints for creating and logging in to users must be implemented by my app. That is to say using these modules is only a class/method library and not an OAuth framework, and makes no calls to an external server.

There are two endpoints used for creating and registering users that receive a username and password (as plaintext because implementing https protocol was infeasible for this assignment) and upon successful login or sign-up will return a token that the front end then maintains and uses as authentication headers for future API calls to create and modify entities which have the user entity as their highest ancestor.

The tokens returned by the server allow for authorization to create and modify entities in the datastore. I chose to use these tokens instead of username and password for both security reasons and for maintaining statelessness of the RESTful server API. Instead of creating user sessions it is required that every call (other than GET calls to publicly visible entities) be authorized and attached to a user; But It would be bad security practices to cache these credentials on the user's device. So instead the front end device maintains a temporary token; allowing the user to remain logged in but maintaining this state on the client device instead of the server.

For storing username and password on the server I made use of the User Model create_user method that creates a user model entity in the datastore and creates a salted hashed password field. I considered implementing my own salted hash scheme because the documentation page for the user model did not specify how the password was being stored. But I investigated the source code of the user model methods and saw that it used the Security module to generate a salted hash for storing password. It is always safer to use existing crypto implementations than to re-implement them.

User Account API

Below are the two endpoints for logging in or signing up with example response codes and messages

URL	Required fields	Response Code	Response MSG
/register	{name:””,password:””} via POST	200	{ "msg": "User Created", "username": "JackABoy", "token": "567itj15EWSyTjOB9eYOOM", "userid": 5636318331666432 }
		400	{ "msg": "Unable to create that user: Username Already Taken" }
		400	{ "msg": "Missing Required Field" }
/login	{name:””,password:””} via POST	200	{ "msg": "User Logged In", "username": "Brandon", "token": "PDnIUpOsyc6Cxoga36hTfl", "userid": 5769015641243648 }
		400	{ "msg": "Missing Required Field" }
		401	{ "msg": "Invalid Password" }
		401	{ "msg": "User Does Not Exist" }
/playlist/<id> /snippet/<id>	{id:##, token:””} Via Request Headers	200	Authentication successful, .json returned, entity modified
		401	{'msg': 'User Validation Failed'}

HTML or JSON GET requests and authentication

The app still maintains the ability to share playlists with other users, who can then view them either in the app or a web browser. But these calls to GET a playlist or a list of all playlists will only return ones that were created as publicly visible, unless authenticated with a token in the request header. So even though private entities have a URL for making GET/POST/PUT/DELETE calls, those URLs are only visible to the user if they have elected to make the playlist public and share it. And unauthenticated requests by a browser or other network traffic will return a 401 error. As an example here are the html and json GET urls for a private playlist that I retrieved from my GAE datastore console that will return an error message
http://swansonbfinalproject.appspot.com/playlist/ahZzfnN3YW5zb25iZmluYWxwcm9qZWN0ciYLEgRVc2VyGICAgIC6xoEKDAsSCFBsYXlsaXN0GICAgIDA35cLDA/
http://swansonbfinalproject.appspot.com/playlist/ahZzfnN3YW5zb25iZmluYWxwcm9qZWN0ciYLEgRVc2VyGICAgIC6xoEKDAsSCFBsYXlsaXN0GICAgIDA35cLDA.json

And here are HTML and JSON links to a public playlist that will render in a browser or return a JSON for rendering
http://swansonbfinalproject.appspot.com/playlist/ahZzfnN3YW5zb25iZmluYWxwcm9qZWN0ciYLEgRVc2VyGICAgIC63J8KDAsSCFBsYXlsaXN0GICAgIDA4coIDA/
http://swansonbfinalproject.appspot.com/playlist/ahZzfnN3YW5zb25iZmluYWxwcm9qZWN0ciYLEgRVc2VyGICAgIC63J8KDAsSCFBsYXlsaXN0GICAgIDA4coIDA.json

Mobile Unique Features of the App

The app makes use of touch gestures that are part of the app interaction vocabulary but unavailable on a desktop browsing (mouse and keyboard) experience. The user can swipe left and right to navigate through a playlist, and long press on items to bring up more options.

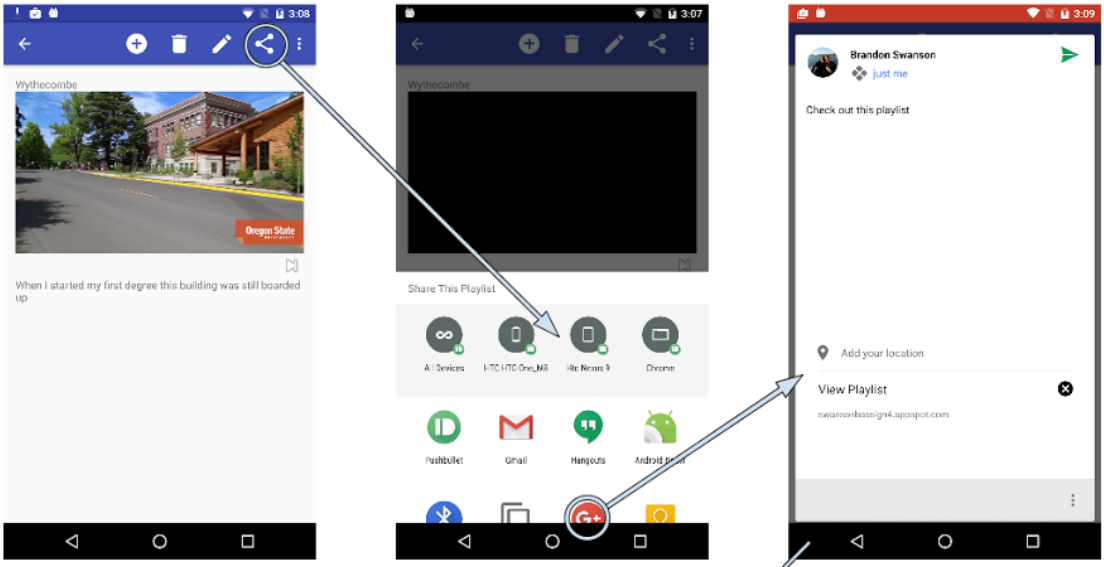
The app makes use of the familiar share icon in the Android eco-system to provide users a way for sharing the content of the app/API with others. It is important to provide this functionality to users in this context as copying the address bar from the browser is not an available action within the app, but users might want to share with others something they are viewing within an app. After touching the share icon from either the main activity or a playlist they are viewing they are given an activity chooser that allows them to select any app on their device that supports sharing of plain text, such as their SMS application, Facebook, “copy to clipboard”, email. My app then sends a plaintext URL corresponding to the url of the entity they are sharing to the app the user selected to use to share this link with others.

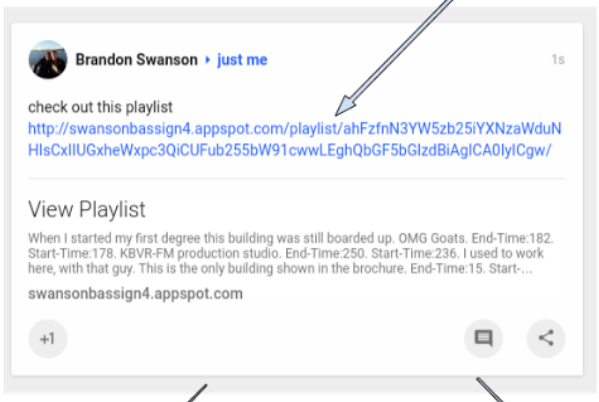
This link that is shared by the app, or a link copied from a user visiting the HTML web page can then be viewed in any of the following ways:

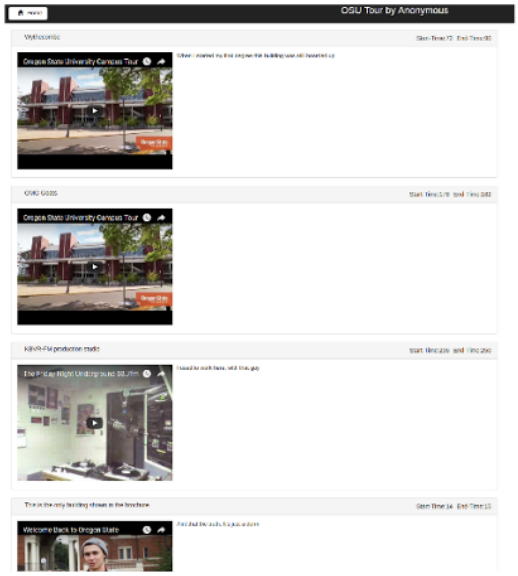
- Desktop/Laptop web browser
- Mobile web browser
- Mobile application

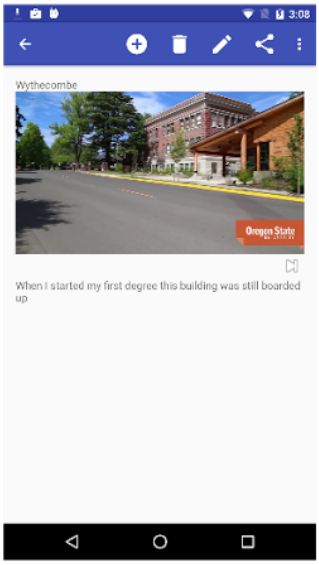
Using an Android feature called Deep Linking when a user that has my app installed clicks on a link to an entity of my API while using that mobile device, for instance in a message from a friend or a social network post, or even a link on a webpage viewed in a mobile browser, they will be able to view that entity represented in the mobile app. For example a link such such as: http://swansonbfinalproject.appspot.com/playlist/ahZzfnN3YW5zb25iZmluYWxwcm9qZWN0ciYLEgRVc2VyGICAgIC63J8KDAsSCFBsYXlsaXN0GICAgIDA4coIDA/
Can be viewed on a computer's browser of launch my app's activity and load the playlists information on a user's mobile device.

See graphic on next page.


3:08
Wythecombe
When I started my first degree this building was still boarded up
3:07
Wythecombe
Share This Playlist
Pushbullet
Gmail
Hangouts
3:09
Brandon Swanson
just me
Check out this playlist
Add your location
View Playlist
swansonbassign4.appspot.com


Brandon Swanson • just me
1s
check out this playlist
http://swansonbassign4.appspot.com/playlist/ahFzfnN3YW5zb25iYXNzaWduNHIsCxIIUGxheWxpc3QiCUFub255bW91cwwLEghQbGF5bGlzdBiAgICA0IyICgw/
View Playlist
When I started my first degree this building was still boarded up. OMG Goats. End-Time:182. Start-Time:178. KBVR-FM production studio. End-Time:250. Start-Time:236. I used to work here, with that guy. This is the only building shown in the brochure. End-Time:15. Start-...
swansonbassign4.appspot.com
+1


OSU Tour by Anonymous


3:08
Wythecombe
When I started my first degree this building was still boarded up

Source Code Highlights

API Test Script

https://github.com/swanyriver/Mobile-Assignment-2/blob/finalProjectAPI/APItest.py

Included in the source code is a python test script that populates the datastore with entities and demonstrates the user authentication protocols.
This script shows examples failed and succeeded calls to login, register, and create or modify entities. There are examples of the server handling calls that are missing required credentials, incorrect credentials, or calls to modify an entity with genuine credentials but not for the user that is the owner of the entity that was requested to be modified or deleted.

Example of Test Script Output

https://github.com/swanyriver/Mobile-Assignment-2/blob/finalProjectAPI/outputAPItest.txt

Network Traffic Module for Android Client

https://github.com/swanyriver/MOBILE-assingment-4/blob/finalProject/app/src/main/java/com/brandonswanson/assignment4/NetworkFetcher.java

All network traffic in the Android app makes use of this class and most calls are done using the classes defined in this file. The classes and interfaces defined allow for instantiating a class that can make repeated calls to a a URL and upon the asynchronous result the UI can be updated using the callback defined in the NetworkFetcher.NetworkFinish interface I defined that provides the network response code and message.

To adapt the Android app to make use of the updated API that required authentication the Credentials singleton class (https://github.com/swanyriver/MOBILE-assingment-4/blob/finalProject/app/src/main/java/com/brandonswanson/assignment4/Credentials.java) was created that stores the returned token credentials and attach them to network call headers.

User Authentication Server Side Code

https://github.com/swanyriver/Mobile-Assignment-2/blob/finalProjectAPI/user.py
Two WebApp2 handlers for registering and logging in, and a function used for validating a user and returning that user key for querying entities owned by the user (owned in that they are the only one allowed to make modifications to that entity and their user key is the ancestor entity for all entities they created, ensuring strong consistency when viewing their entities)