Final Project

SinClair Fuh

David Opie

Password Saver

**1, Overview:**

This final project builds off Sinclair Fuh’s implementation of a user-login system specified in Assignment 3. Previously, logging in using authenticated credentials would merely show a toast to the user. Now, logging in using authenticated credentials would take the user to a page where they can submit credentials to other websites to store in a DynamoDB database, and display all credentials the user stored so far. There are two main activities visible upon logging in:

* Loggedin.java: an activity that prompts the user for a set of credentials to store in DynamoDB. Contains a button that takes them to userinfo.java.
* Userinfo.java: an activity that displays all credentials submitted by the user so far. Contains a button that takes the user back to Loggedin.java.

In addition, shaking the phone hard enough while the user is logged in would log them back out, prompting them to re-enter their credentials again. This was accomplished using accelerometer data, prompting AWS amplify to sign the current user out and return them to the login screen. Note that how hard the user needs to shake the phone is defined by a variable called **shakeThreshold.** This variable is present in both userinfo.java and loggedin.java, and is initialized to 10f. Feel free to experiment with this variable, though we’ve found that setting it to anything under 10 would instantly log out the user upon them logging in due to how sensitive it causes the app to be to any screen vibrations.

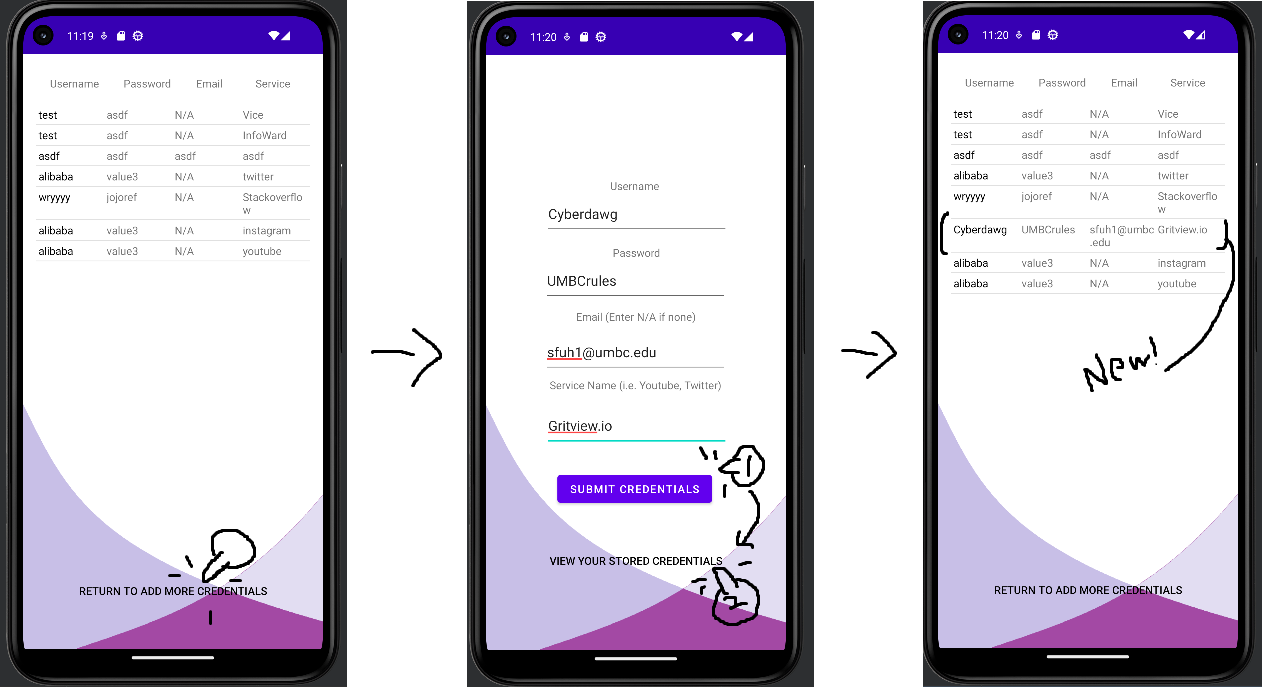


Figure : Example workflow of our password saver for a user. Screens 1 and 3 are userinfo.java, and screen 2 is loggedin.java.

To run this code, start by registering yourself as a user using the provided interface, then log in to begin using the password saver component. Note that when signing up, passwords have to follow the following specifications:

* Must be 8 characters or longer.
* Must include at least 1 special character + 1 numerical character
* Must make use of uppercase and lowercase letters.

I’ve also registered an account with the following credentials for you to use:

* Username: praise\_nilanjan
* Password: 1@3$UmBc

**2, Amazon API Structure:**

The password saver is an application that uses android studio and AWS to store users passwords they use for different services. This project can be roughly separated into two parts: the authentication service and the password saver itself. First, the authentication service uses android studio and AWS Cognito to protect and store user data. The authentication service relies on AWS Amplify library to securely store and manage user passwords. The password saver itself uses android studio to collect the passwords from users, stores them in AWS’s service DynamoDB, and once signed in allows users to view all saved passwords including the username, password, email address, and service name associated with each service. For example, Spotify’s saved password would look like {spotifyUser, wordpass1, [thebeststudent@umbc.edu](mailto:thebeststudent@umbc.edu), Spotify}.

AWS’s usage with DynamoDB is separated into 3 parts. AWS’s API Gateway provides us with a HTTP link. Providing that to Android Studio we can connect to different Lambda functions. There are lambda functions for each method we use with DynamoDB; PUT, GET, etc. The DynamoDB is a NoSQL database in AWS that allows us to store our information in the AWS cloud.

**3, Lambda Functions Used:**

A, AddCredentials: used in the activity loggedin.java to submit user-provided credentials to their DynamoDB entries.

def lambda\_handler(event, context):

result = {"code": -1, "message": "placeholder"}

owner = event['owner']

uname = event['uname']

pword = event['pword']

email = event['email']

sname = event['sname']

dynamodbtable = client.Table('CredentialInfo')

for element in (owner, uname, pword, email, sname):

if len(element) == 0:

# result["code"] = 401

# result["message"] = "Error: field left blank."

set\_result(result, 401, "Error: field left blank.")

return result

dbkey = owner + "-" + sname

if dynamodbtable.get\_item(Key= {'Owner': dbkey}).get("Item") != None:

# result["code"] = 402

# result["message"] = "Error: account with that username already exists!"

set\_result(result, 402, "Error: account with that username already exists!")

return result

resp = dynamodbtable.put\_item(Item = {'Owner': dbkey, 'Uname': uname, 'Pword': pword, "Email": email, "Sname": sname})

set\_result(result,

B, RetrieveDBEntries: used in the activity userinfo.java to retrieve and display all stored credentials in DynamoDB for the logged in user.

import json

import boto3

from boto3.dynamodb.conditions import Key, Attr

client = boto3.resource('dynamodb')

def set\_result(json, code, message):

json["code"] = code

json["message"] = message

def lambda\_handler(event, context):

result = {"code": -1, "message": "placeholder"}

owner = event['owner']

dynamodbtable = client.Table('CredentialInfo')

if len(owner) == 0:

set\_result(result, 401, "Error: field left blank.")

return result

set\_result(result, 200, json.dumps(dynamodbtable.scan(

FilterExpression=Attr('Owner').contains(owner)

)["Items"]))

# TODO implement

return result

**4, Ways this project can be improved on:**

* Clipboard-copy capabilities: The ability for users to tap a certain field in userinfo.java (the activity that displays all their credentials) and copy that information to their clipboard.
* Swipe-to-remove credential capabilities: The ability for users to swipe a certain entry within userinfo.java and remove that entry from the database of their stored credentials.
* Proximity-based screen blackouts: A setting for the app to automatically obscure its own screen when it detects itself to be in close proximity to another phone.