**Tempo:**

AI driven Image-to-music Social Media Platform

Video Demo: <https://www.youtube.com/watch?v=epbASCz5dwc&feature=youtu.be>

Github Repo: <https://github.com/jerry-z/tempo_app>

Jerry Zhang, UNI: jz2966

Wenhao Li, UNI: wl655

COMS6998: Big Data and Cloud Computing

Professor Sambit Sahu

**ABSTRACT**

Utilizing AWS’s microservices and serverless architecture, we aimed to develop a unique social media app—*tempo*, that takes advantage of both novel advancements in cloud computing and machine learning. With tempo, users are able to create, share, and save storyboards—image, music, and content driven posts. The app offers a platform for users to explore their own creativity as well as take advantage of AI technologies in image classification and NLP to match songs to their respective storyboards.

**INTRODUCTION**

Many social media apps such as Snapchat, Instagram, and Tiktok have invaded the phones of Millennials and Generation Z, taking up millions of hours of our spare time. Users have found solace and enjoyment in sharing images, stories, and messages with their friends, families, and social networks.

We aimed to inject the latest innovations in AI, specifically in image classification and NLP, into the social media space with an app that allows users to post and send photos that are coupled with music options generated by our AI-powered backend microservices and algorithms. This will allow for an enjoyable way for users to express emotions derived from their current mood as well as the sentiments and topics found in their images. Using AWS’s internal machine learning image classification microservice, recognition, as well as writing our own NLP functions allowed us to create the backend technology. The entire system architecture, however, is entirely predicated and founded upon the serverless architecture and microservices that AWS provides, which allows us to utilize their cloud computing platform to host and run the app.

When a user signs up for a profile, they are able to access our application’s home portal, where they are greeted with four unique tabs at the bottom of the page: 1. Home, 2. Search, 3. Storyboard, 4. User Profile. The home page provides a newsfeed in which the user can view, comment, like, and save the posts of other users that they follow. The search page is used to both search music as well as other users, depending on which navigation bar is selected. After querying, users can follow users and save songs to their profile from this page. The storyboard page is where the heart of the application lies, as it allows users to create storyboards in which they can share music, posts, and images. Additionally, the application allows them to user our AI backend to generate music results based on their uploaded image and caption text. By allowing users to interact with one another on the newsfeed through likes, bookmarks, and saved music, we want to imbue our userbase with an interactive, creative, and unique culture.

**ARCHITECTURE**

The following figure illustrates the application’s architecture and how the AWS services and Spotify API are integrated with our frontend that is hosted on S3.

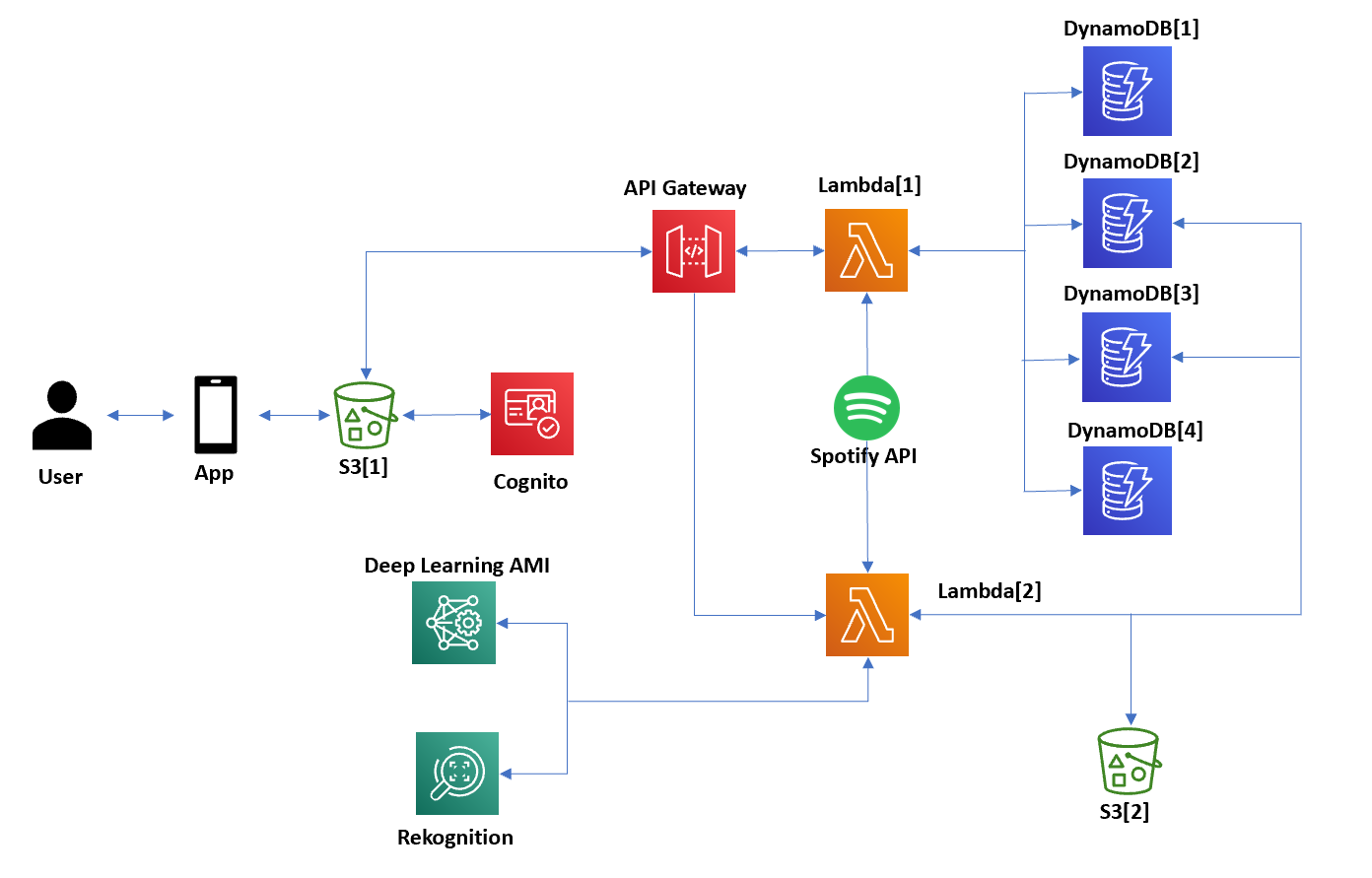


Figure 1: Tempo’s architecture

**IMPLEMENTATION**

*S3*

Our frontend web app, which is written in html, css and js is hosted in an S3 bucket. Additionally, we have S3 buckets dedicated to storing user profile pictures, and story photos.

*Lambda Functions*

The app currently utilizes nine lambda functions that are all invoked over HTTPS by our API gateway. All the lambda functions serve to handle the API request made by our users. The lambda functions serve the following roles:

Lambda[1] get\_newsfeed: This function is called every time the user loads or reloads their home or profile pages. With the function, depending on its input requests, it can return a response to get the entire public newsfeed, get the newsfeed of a single user, get the newsfeed of a single user and their respective friends, and get saved newsfeed items for a single user. This function allows the app to dynamically load and update storyboard interaction between users.

Lambda[2] log\_sign: This lambda function serves as a temporary but effective substitute for Cognito. With this function, we can both use it to sign new users up or log a user in depending the request body parameters.

Lambda[3] get\_music: This lambda function is able to search and return a corpus of music based on input search text. Additionally, it can be used to save songs to a user’s profile as well as get the favorite songs of an individual user for user profile robustness.

Lambda[4] upload\_binary: This lambda function is used to transform an uploaded image into a binary file and into an S3 bucket for later usage as either a profile picture or as a image that may need to be passed into AWS Rekognition.

Lambda[5] upload\_story: This lambda function uploads all the necessary information from a story into a DynamoDB database (music url, name, user, description, etc) with a key equal to the timestamp a story is posted, thus allowing correct newsfeed loading when Lambda[1] is called.

Lambda[6] edit\_user: Using this function, the app is able to edit user profiles and change information such as bios, friends, saved music, and saved stories. It can also be called when a friend wants to be added or removed from a user profile.

Lambda[7] get\_user: With this function, the app is able to get a user’s specific profile and all their information (saved posts, saved music, posts, friends lists). Additionally, this function can be called to search for users by key strings,

Lambda[8] edit\_story: This function is specifically used to edit details about a particular story a user may want to change. Attributes of a story such as changing a caption, adding a like/bookmark, and adding a comment utilize this function. With a different body request, this lambda function also serves to delete stories entirely from a user profile and from the app in general.

Lambda[9] ai\_search: This function is connected to and calls AWS rekognition as well as AWS comprehend to create our AI backend for music recommendation. This function is called when a user is making a storyboard and wants music recommendations based on their image and captions. By getting a list of key labels from these services called from this function, the lambda function further sends a request to Spotify API to get the final list of recommended songs to a user.

*Databases*

The system architecture for our application utilizes two DynamoDB databases in order to store all the necessary data: 1. UserId, 2. StoryId. The UserId database keeps all user info including login info, email, bio, friendslist, profile picture s3 address, collection list, comments list, favorited music, and liked list. The purpose of the second database, StoryId, is to be able to keep a detailed account of the likes, saves, and comments for each story as well as the image S3 endpoint, music data (artist, song, album, spotify webplayer url).

*API*

Rekognition: We use this API in order to take advantage of facial analysis, text recognition, and object detection from a user’s posts, which then can be used to extract relevant labels for AI music search.

Comprehend: The app utilizes this API in order to perform sentiment analysis, syntax detection, and feature extraction from a user’s story’s captions, and like Rekognition, can be used to call the Spotify API and return relevant music results.

Spotify: The app utilizes the Spotify API in order to return songs based on user inputs for our search and storyboard pages.

*Frontend*

The entire application was written in .html, .js, and css, which allowed us to create a web application. It is currently being hosted in an S3 bucket. Utilizing Bootstrap and Font Awesome libraries, we were able to easily implement icons and features that vastly improved our UI.

**RESULTS**

After linking the frontend web application with the serverless backend, we were able to produce the following results

.

*Login*

The following figure shows the user flow from signup to login.



Figure 2: Login/Signup pages

Originally, this app was going to use Cognito for user signup, login, and verification; however, we were unable to implement this microservice given the time and focused on developing the other aspects of the project. All the data from sign in is stored in a the UserId DynamoDB. Additionally, the log\_sign lambda function is utilized for both signup and login to store info and direct users to their custom profiles depending on their login credentials.

*Newsfeed*

On the newsfeed, users will be able to scroll through and view the storyboards of themselves and the other users they may follow in the order that they were posted. Additionally, with each storyboard, users can like, bookmark, and comment on them. All the aforementioned options will be stored within the StoryId DynamoDB.

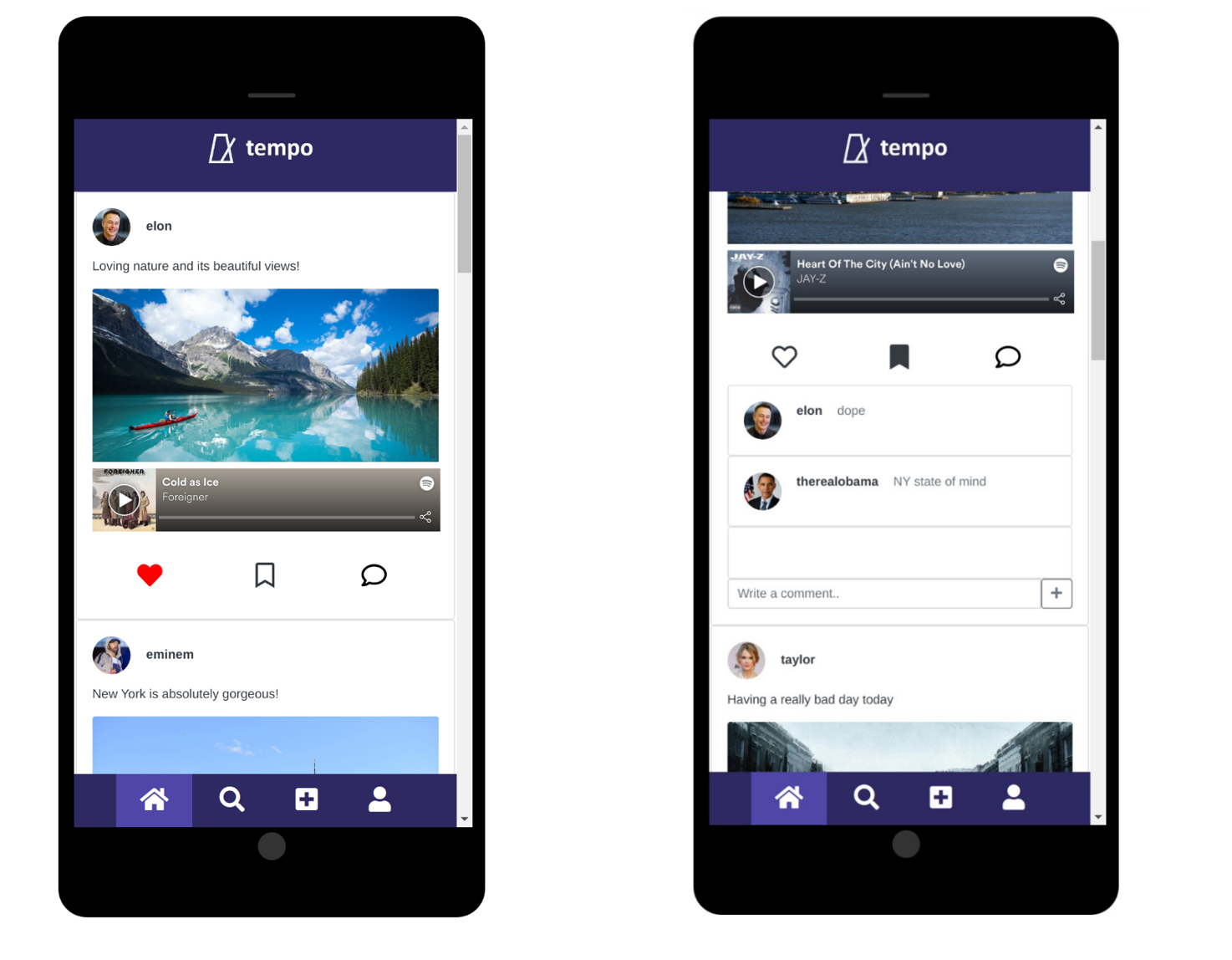


Figure 3: Newsfeed Pages

Pressing the comment button will also open a navigation bar that shows other user’s previous comments and allow users to post their own.

*Search*

The following figure shows the two main navigation sections of the search page.

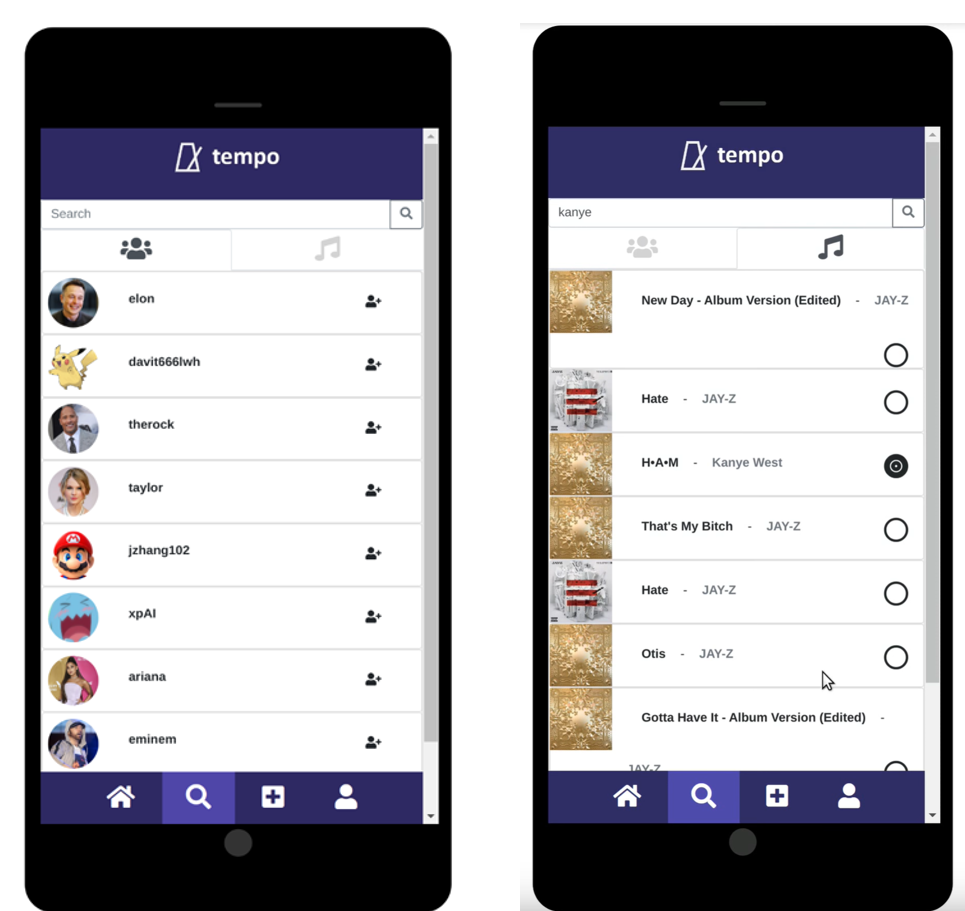
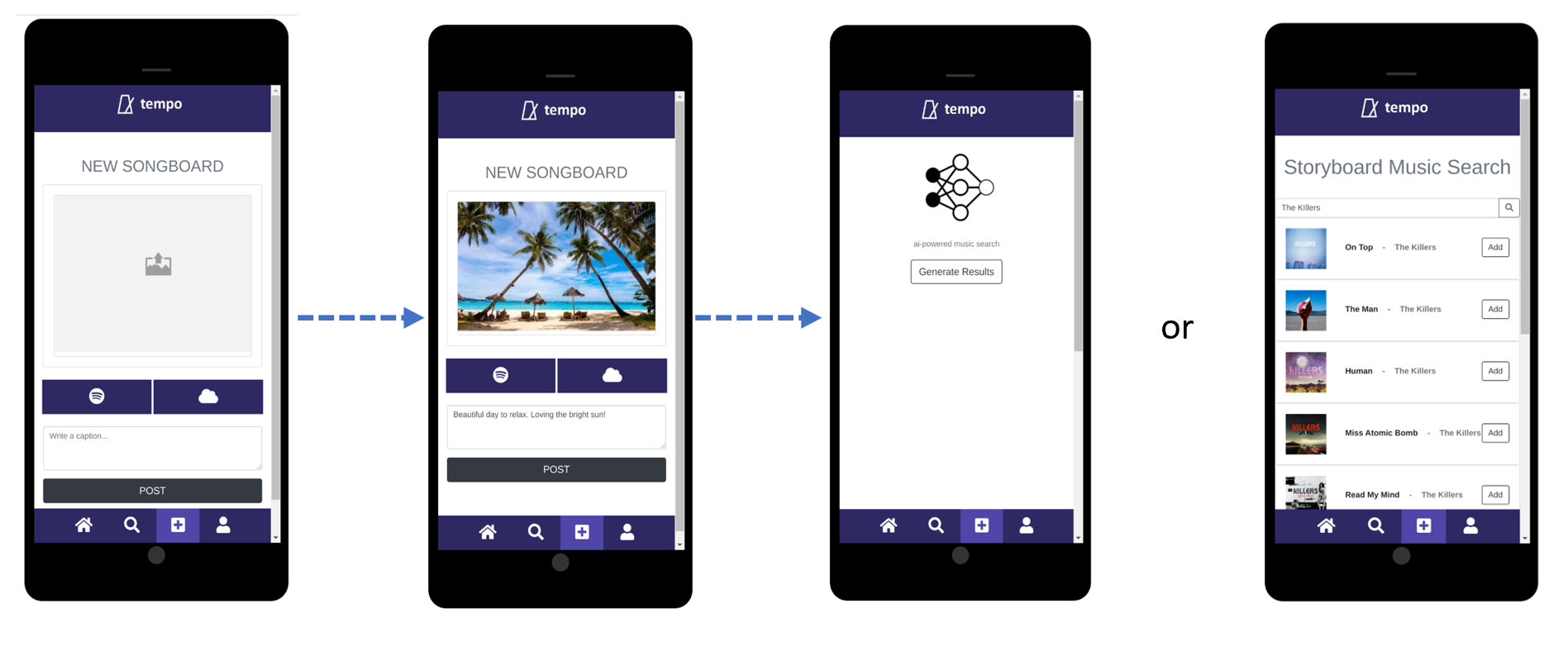


Figure 4: Search

The first navigation tab allows users to search for other users on the platform and subsequently follow them in order to see their posts. The second navigation tab allows users to look up music via the Spotify API and then also save songs to their saved song section in their user page.

*Storyboard*

The following images show the user flow from upload to storyboard/ai search.



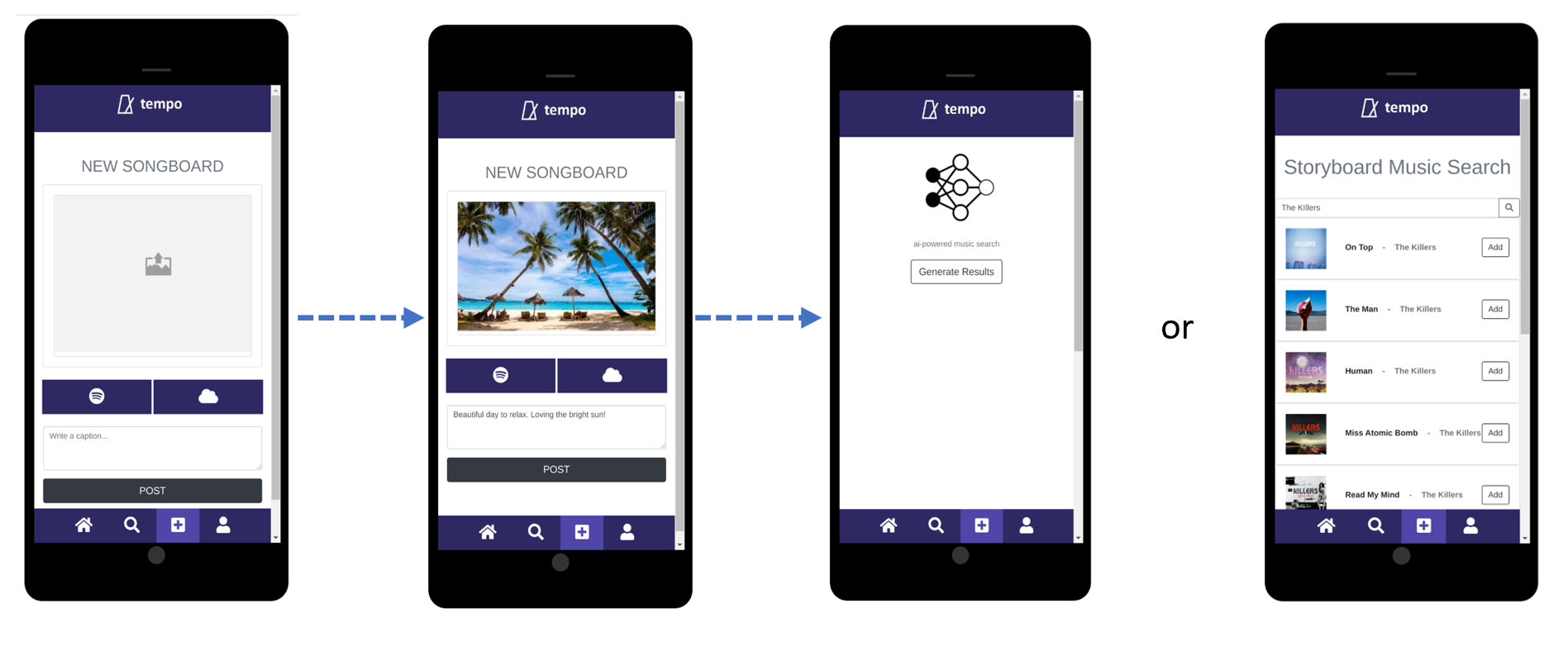
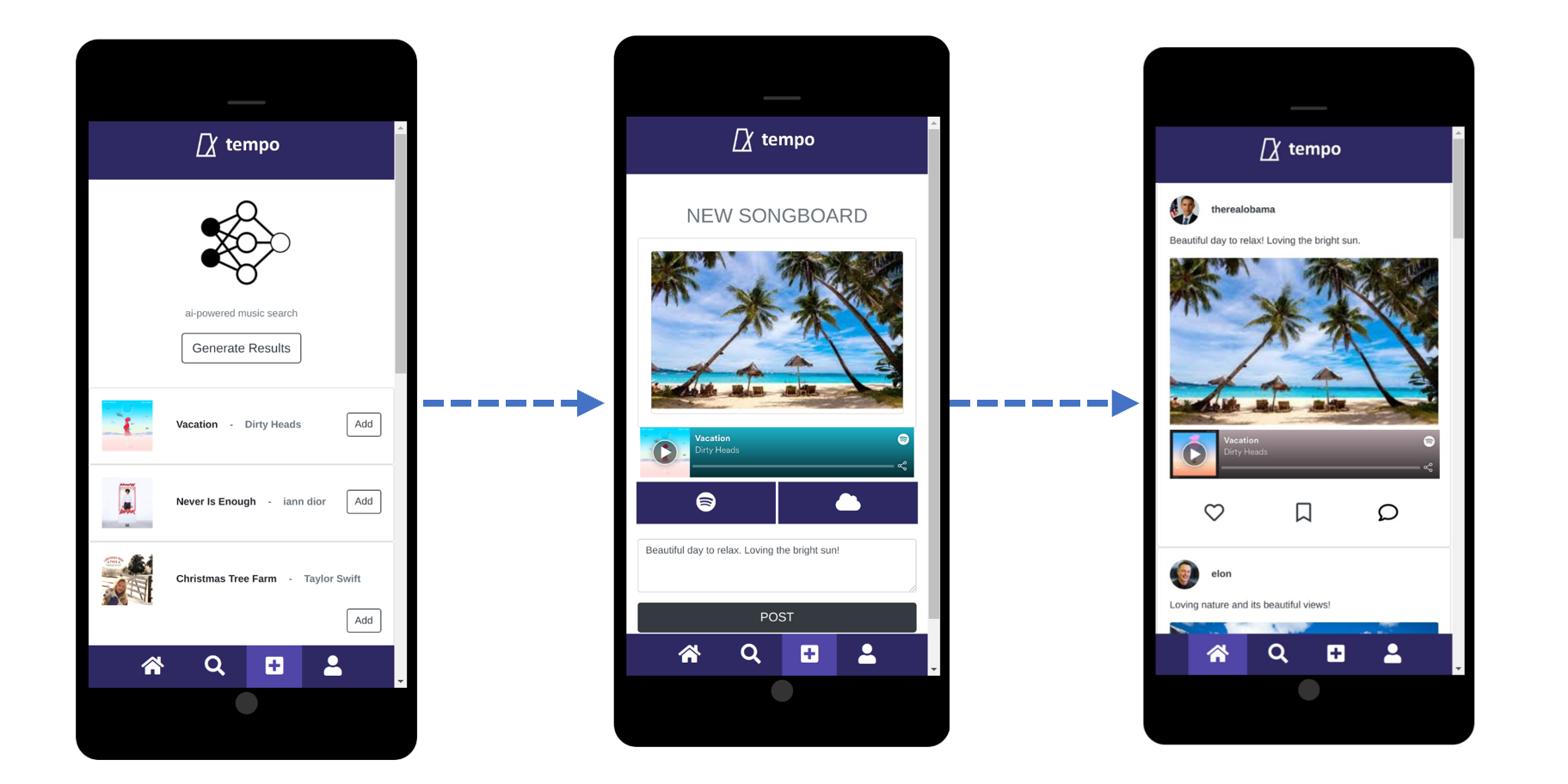


Figure 5: Storyboard upload user flow

From the upload page, users can create and customize their own storyboard with whatever image, caption, and music they’d like. They can add their own desired music via the spotify button or add music found via our rekognition/comprehend backend.

The next figure shows specifically the user flow from ai song generation to post.



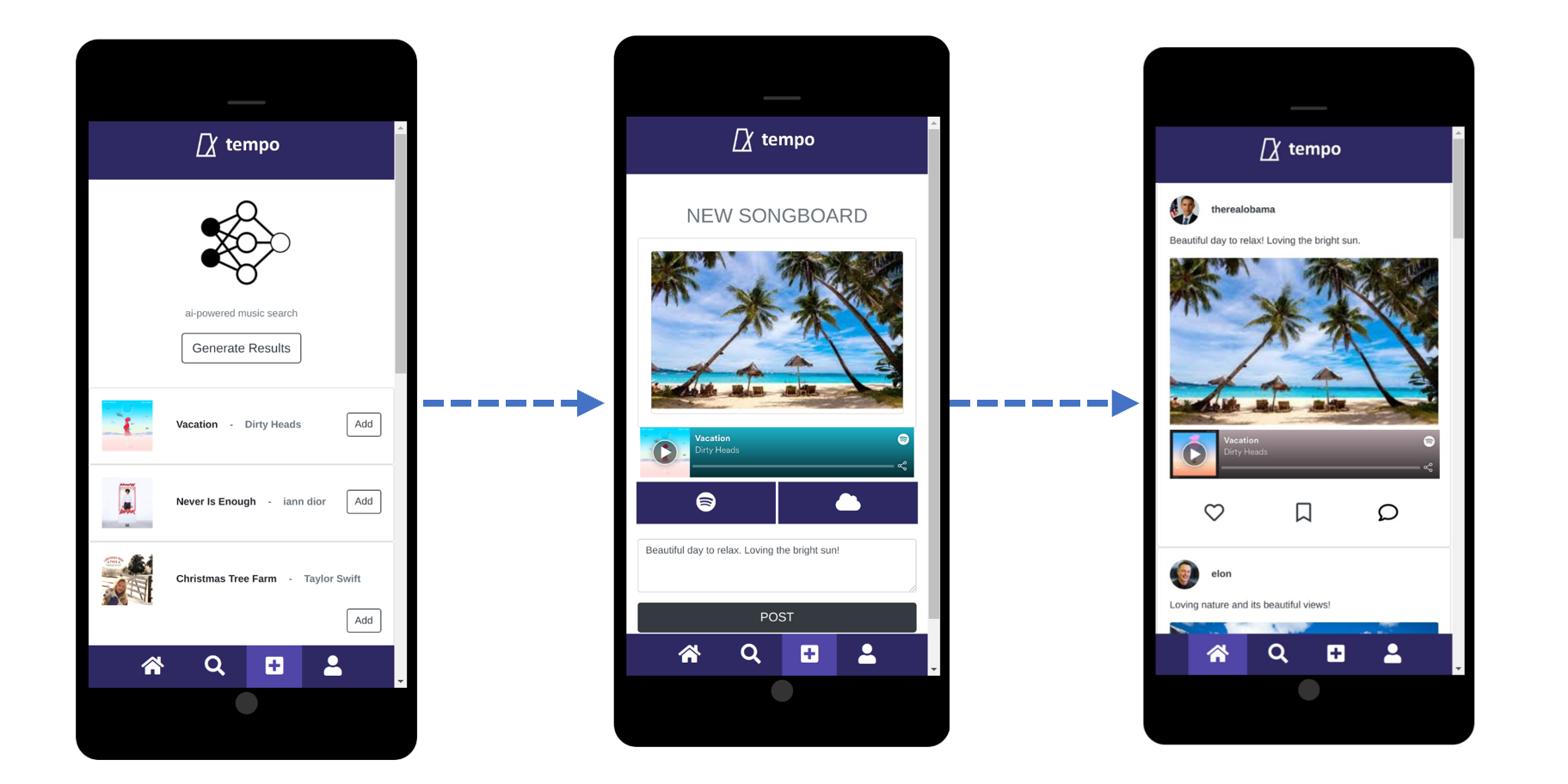


Figure 6: Storyboard Post user flow

Note that the rekognition backend utilizes the image as the main format the extract labels while comprehend uses the text currently within the caption to extract relevant labels. Together, we have a corpus of labels that we search and append songs to within the music search for the AI music search

*User Profile*

The last navigation bar is the user profile page, which is shown below.

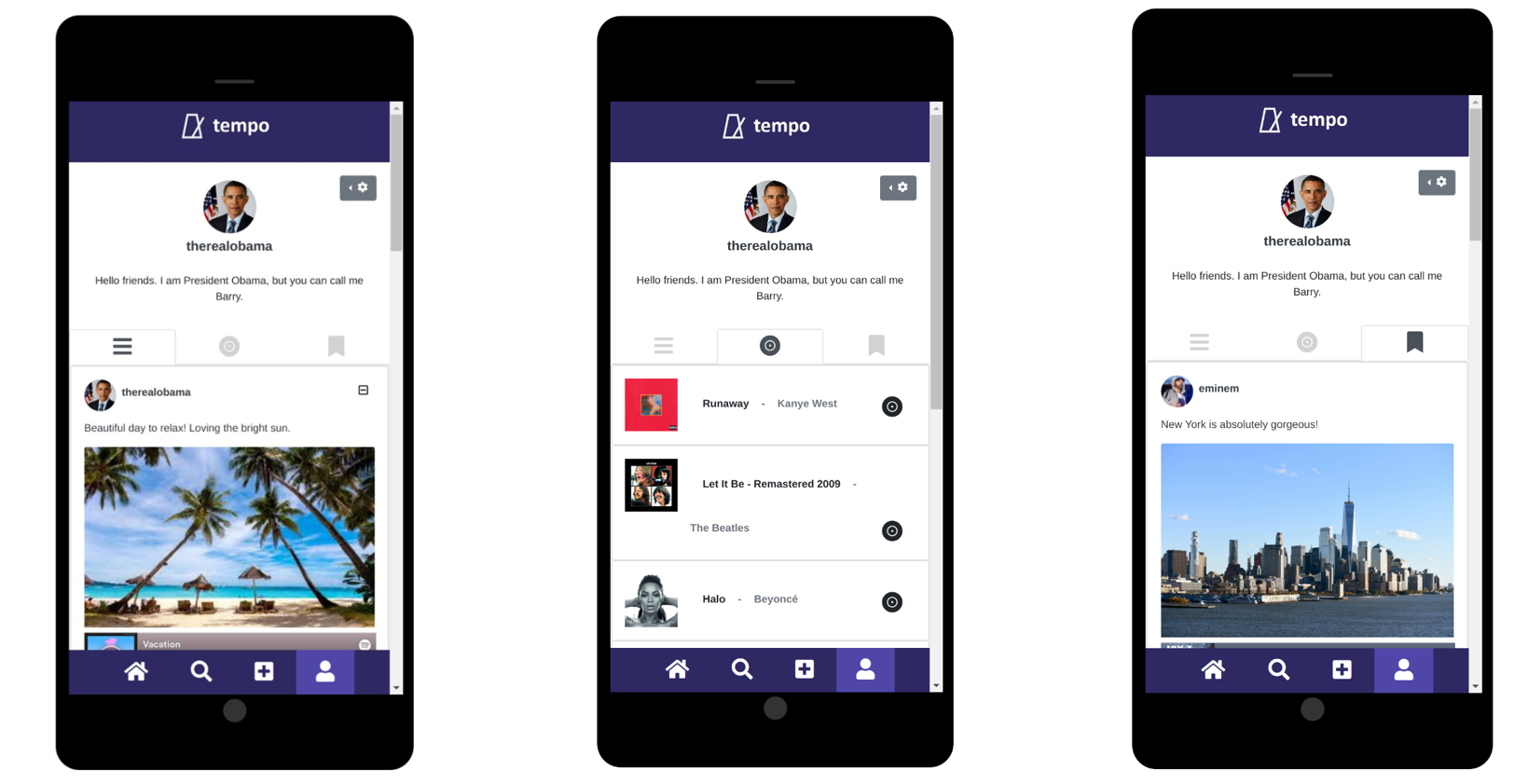


Figure 7: User Profile Navigation Tabs: a. user storyboards, b. user saved music, c. bookmarked storyboards

From this page users can view or delete their own posts, view their saved music, and view their saved posts.

**CONCLUSION + NEXT STEPS**

Unfortunately, due to the limited time constraint of the semester as well as our group only having two members, we were unable to completely add all the features and versatility that we aspired to at the beginning of the project.

While AWS does offer a host of microservices designed for scalability, our front-end codebase is not yet designed for a massive userbase. Key functions and tasks that we would have liked to accomplish given that we had more time are: cognito integration, infinite scrolling, playlist creation, more robust music hosting, conversion from a web app to a hybrid app, and a more robust AI-backend.

To that end, we do feel that we were able to accomplish a large assortment of tasks and end up with a very functional, serverless app that taught us a whole host of knowledge regarding the systematic workflow required to create scalable and serverless technologies on the cloud.