Project Report: TattScan

Joey Tatú  
15015556  
Cloud Platform Programming  
Postgraduate Diploma in Science in Cloud ComputingNational College of Ireland  
joey.tatu@student.ncirl.ie  
16th December 2021

*Abstract*—–The project has been created using Python coding in Amazon Web Services (AWS) Cloud 9 IDE using AWS S3, DynamoDB, Rekognition, Elastic Beanstalk. The purpose of this project is to create a Flask app to use image recognition technology to scan a person’s tattoo, analyse it and search for similar photos on Twitter using a generated hashtag. Image recognition software are quite able to scan tattoos, but not specific elements of a tattoo. This is the main focus of the project. Other features allow users to upload and save photos in the S3 bucket, download a file, and search for similar photos on Twitter. AWS Elastic Beanstalk was used to deploy the webapp, which came with many issues including issues with Elastic Beanstalk not recognising a Python library or its version number along with the package manager needing to be updated, but not being allowed to, possibility resulting from permission issues with AWS Educate.

# Introduction

## Background

The inspiration for this project originated from the creation of a vegan health and fitness app. This project's concept was to take a snapshot or image of the eyes and provide a result depending on the sclera's whiteness or yellowness. Against support of the original concept, this application was expanded into an Android app that scans for diabetic retinopathy in the eye. [1] This project has a similar concept where a user can upload a photo of a tattoo and have the application identify it.

## Aims

Google Lens is a comparable image recognition system that can recognise a tattoo but not the pieces that make up a tattoo. Barcodes, QR codes, labels, text, famous buildings, and regions are all recognized by Google Lens. [2] [3]

TattScan works in a similar way to the diabetic retinopathy app. TattScan analyses and identifies a tattoo's primary elements, then develops a hashtag that it uses to search Twitter for comparable tattoos, which it then displays its results to the user. The user can decide if they wish to save the result.

This project has aimed to build a bridge between Google Lens and identifying elements of tattoos via image recognition. Currently, photo identification technology can identify tattoos, but not elements of a tattoo. This project aims to rectify this.

## Technology

TattScan is a web application that has been created using the Amazon Web Services (AWS) Cloud 9 integrated development environment (IDE), primarily using Python code. Possible future iterations will be developed for Android devices.

It was expected that Django would be used as a Python web framework. Comparably, a web site can be hard-coded, but tools are available to simplify this. This is like Flask which has been used instead. Flask removes the bulk, hard-coding aspect of a web framework and simplifies creating and deploying a web application. [4]

AWS Simple Storage Service (S3) has been used to store data files (objects) in containers (buckets). [5]

AWS DynamoDB has been used in thus project along with S3. AWS DynamoDB is a NoSQL database that remains consistent even when the database becomes larger or smaller. DynamoDB can also handle vast amounts of semi-structured data while adhering to rigorous latency constraints. [6]

These are the main aspects of this project in relation to storage. Further information on cloud services used in this project are in the Cloud-based Services section.

# Project Requirements

## Functional Requirements

The functional needs are ranked in this section. These describe the goals of the project.

### Requirement 1: Identify tattoo

This use case describes how a user uploads an image of a tattoo.

#### Description & Priority

This is the focus of this project. This use case explains how a user uploads an image of a tattoo and how the system identifies it. An option then is to allow the user to search the image on Twitter by using a hashtag.

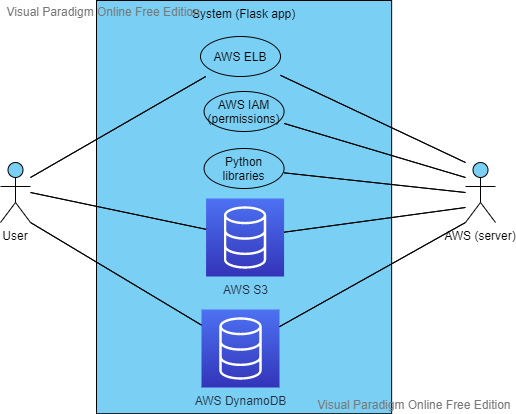
#### Use Case

Unique ID: identify-tattoo

**Scope**

The scope of this use case is for a user to upload an image and for the system to identify it as a tattoo and its main elements.

**Use Case Diagram**



(Larger images are at end of report for easy reading)

**Flow Description**

**Precondition**

* AWS and its services are active and running correctly.
* The Flask webapp is coded correctly and connected to the relevant services (e.g., AWS S3)
* Twitter and its API are active and running.

**Activation**

This use case begins when a user opens the Flask webapp.

**Main flow**

1. The user opens the Flask webapp.
2. The page loads with options to sign up, login and to upload a file. <See A1, A2>
3. The user selects upload a file

(External: User selects image from their device)

1. Image is stored temporarily in AWS S3
2. The relevant Python library identifies the image
3. The output if what it has identifies is outputted to the user.
4. The User clicks on one of the outputs. <See A3>
5. The System prefixes the output with a hash symbol and adds “tattoo” to the output to create a hashtag. (e.g. #starstattoo)
6. Using the Twitter API via a Python library, the System searches Twitter for images matching the generated hashtag.
7. The Systems then displays the results to the User.
8. The User clicks on an image   
   (External: User is taken to Twitter website)

**Alternative flow**

A1 <User logs in>:

1. The User clicks log in
2. The User enters their username and password
3. The System queries the DynamoDB to check credentials <See E1>
4. The User successfully logged in

<returns to Step 3 in Main flow>

A2 <User signs up>:

1. The User clicks sign up
2. The User enters their username and their password and their password again to confirm
3. The System create a new entry in the DynamoDB and encrypts User’s password.
4. The System logs in the User
5. The User successfully logged in

<returns to Step 3 in Main flow>

A3 <User doesn’t wish to search Twitter>:

1. The User decides not to click on an output.

<Use Case ends>

**Exceptional flow**

E1 < User’s username and/or password are incorrect>

1. The System outputs that the username and/or password are incorrect to the user.

<returns to Step 3 in Alternative flow: A1>

**Termination**

This use case is terminated when the User selects an output to go to the Twitter website.

**Post condition**

The system goes into a wait state.

## Non-functional requirements

### Data Requirements

Encryption is required for all data transmissions. It is not permissible to use raw data for communication from the Flask webapp to a service or Twitter API.

Passwords must contain at least one uppercase letter, one lowercase letter, one number, and one special character and must be at least eight characters long. There is a verification in place for this.

### User Requirements

Users should have a basic knowledge of browsing and searching online and uploading files to a website.

### Usability Requirements

The Flask webapp will be easy and straightforward to use. Accessibility features have been implemented such as being able to invert colours of the website to make it easier to read, as well as allowing the text to be made larger or smaller to aid those with visual impairments.

# Architectural Design Aspects

# 

(Larger images are at end of report for easy reading)

The Flask webapp may not appear to be very extensive at first glance but is detailed sub rosa.

AWS was chosen instead of other cloud platforms is because of AWS’s locations among other reasons. If one location fails, another one can be used. There are no restrictions on regional borders and one can use AWS from anywhere in the world. AWS also has a “disaster recovery system” and backup capabilities in different areas. Scalability is another important factor. AWS Auto-Scaling and AWS Elastic Load Balancing allow developers to scale their applications to their needs. [7]

AWS Identity and Access Management (IAM) is used for deployment of the Flask webapp to allow the webapp to be deployed and to monitor its health. AWS Elastic Beanstalk (ELB) is a solution for launching and scaling web apps and services. With using AWS, this was a logical solution to launch TattScan. [8] AWS Cloud9 is an IDE where the coding was completed. Creating the DynamoDB and AWS Simple Storage Service (S3) databases also were completed in Cloud9.

AWS Simple Queue Service (SQS) has been used in TattScan to ensure the server is not overloaded when a large group of people are using it.

Datadog has been used to monitor and analyse performance of TattScan. This is used to monitor whether more or less resources are needed.

TensorFlow is a Google service that is used for image detection. TensorFlow was originally created to be used internally within Google but later was released to the public. TensorFlow can be used to train the algorithm to identify features of images, using large set of images. [9]

TensorFlow also has a Python library but was too large in size to be used in this project.

Instead, AWS Rekognition was utilised. Rekognition is an Amazon cloud service that can recognise objects and faces in photos, including celebrities, as well as facial expressions and emotions. Rekognition can also track a person's walking activity and recognise videos. In a photo, Rekognition may be designed to recognise sexually suggestive content and nudity. This might be handy for websites who wish to keep an eye on explicit media. [10] [11] [12]

Tweepy is a Python library that connects to Twitter via its API service. Tweepy can be developed for use of retrieving, making and deleting Tweets, Twitter searches, Twitter Lists, and other Twitter use cases. [13]

# Cloud-based Services

Not shown in the diagram above is the AWS Elastic Compute Cloud (EC2) instances. The AWS Elastic Beanstalk, AWS Cloud9 and AWS Simple Storage Service (S3) all are an individual EC2 instance. Apps deployed to the AWS Elastic Beanstalk can make use of these services. It would be redundant to mention EC2 as a separate cloud service.

Two types of cloud databases have been used: AWS DynamoDB and AWS S3. DynamoDB is a NoSQL database with tables designed for storing structured data. [14] Therefore, it was chosen to store the account information. Because S3 can tolerate a high number of queries and unusual latency patterns, it was chosen to store the uploaded images. [14]

Technical experts and developers utilise AWS SQS to transmit, store, and retrieve many messages of varying sizes asynchronously. [15]

AWS SNS is an automatic service for messaging for communication between applications and people. [16]

Datadog is an online cloud service that is for monitoring and an analytics platform for determining performance metrics, as well as event tracking for infrastructure and cloud services. Servers, databases, and tools may all be monitored with the software. [17]

Datadog was due to be implemented, but there was not enough time to connect it.

AWS Rekognition is a cloud-based service form Amazon that slows the use of image identification, among other features. [10] [11] [12] In this project, a photo of a tattoo is uploaded to the S3 bucket. Then this photo or another is called from the S3 bucket and is passed to Rekognition. Rekognition analyses the image and detects with a degree of confidence what the image contains. Rekognition returns many “name” values and “confidence” values. This “name” is what Rekognition thinks it contains and the “confidence” is how confident Rekognition is.

# Library Description

After this, the “name” is sent to Tweepy, prefixed with a hash symbol and with the suffix “tattoo” to make (e.g.) “#dogtattoo” where “dog” would be the name value from Rekognition. Tweepy is a Python library for use with Twitter’s API. [13] Tweepy then searches media on Twitter and returns media that matches the Rekognition name. These images are posted to the webapp where a user can click on them to go to that specific photo on the Twitter website.

# Implementation

## Flask webapp

A basic flask app was created in AWS Cloud9 to begin with, using the Flask Python library. The main app routes (pages, for simplicity terms) that were to be used were then created. The Flask app when then run in Cloud9 to ensure it could launch correctly. An upload feature was created for S3.

## AWS S3

The S3 bucket that would be storing the images of tattoos was then generated by using tutorials made available and online. After this the S3 bucked was coded into the project where random images found online were uploaded to check they could be accessed from S3 and on the webapp.

## Accounts - SQLAlchemy

After creating the S3 bucker, the accounts section was coded and it was attempted that a python library, SQLAlchemy, would be used to store the account information. This became more troublesome than it was worth, and an alternative was acquired.

## Accounts – AWS DynamoDB

DynamoDB, a semi-structured cloud-based database, was then chosen to store the account information. Coding ensued and testing began before being added to the website. Test data was also trialled on the webapp.

## Python libraries worth mentioning

Some of the Python libraries used in this project were Boto3, Flask, Flask\_SQLAlchemy, SQLAlchemy, Tweepy.

### Boto3

Boto3 is a software development kit that simplifies the process of creating software. It's used to establish, configure, and administer AWS services. [18] For this project, Boto3 was used to connect the project to the S3, Rekognition and DynamoDB services.

### Flask Flask-Bootstrap and Flask-SQLAlchemy

Flask is a microframework that has been written in Python. It does not need the use of any specific tools or libraries. It relies on third-party libraries and software for databases, form validation and to accomplish similar tasks but extensions for Flask can also be used. [19]

Extensions such as Flask-Bootstrap and Flask-SQLAlchemy are used to extend Flask’s functionalities. These extensions act as if they are part of the Flask library. [19]

Flask-Bootstrap is a way to help “beautify” and add functionality to a web app by using code that would otherwise be utilised through a CSS or JavaScript file. Flask-SQLAlchemy was going to be used as the database for the accounts, but this was later changed to use AWS DynamoDB instead. [19]

# Integration, Delivery and Deployment

## Version history & video presentation

GitHub through Git Bash has been used for versioning. The GitHub repository can be found at: <https://github.com/JoeyTatu/CloudPlatformProgrammingProject> (Git log as screenshot below)

Video presentation: <https://youtu.be/gYajeY7rJVY>

## Deployment

The Flask webapp was deployed to AWS Elastic Beanstalk and can be found at: <http://tattscan6-env.eba-xwzppmtg.us-east-1.elasticbeanstalk.com/>. However, the actual app is not available to the issues mentioned below and the webapp only shows “Index of /”.

There were many issues that were faced during deployment. Some versions of Python libraries were not compatible with the Elastic Beanstalk. Over 25 attempts to deploy the Flask app to the Elastic Beanstalk failed (some are below, screenshot). One attempt was to remove the version numbers from the Python libraries in the Requirements file (all required Python libraries for webapp to run). While this fixed the Python library version issue, some Python libraries were not recognised, and alternatives were required. These also required testing by terminating the previous deployed environment and deploying a new one.

With all the coding correct, the deployment failed due to “pip”, a package manager, needing to be updated. However, privileges restricted by AWS Educate did not allow access to the command shell on the webapp, so the app would fail regardless of what was done. At a guess, it would be speculated that two full days of twelve hours each were used to attempt deployment of the webapp.

# Conclusions

It’s very simple to say that if this project was to be restarted or redone, AWS would not be a first choice. It’s very overly complicated and it feels like one needs a Doctorate specifically in AWS just to use it. Granted, there are tutorials and documentation available, but these don’t necessarily make things any way easier. One could get bogged down by the sheer number of services and amount of documentation to go along with these services that nothing may get completed and can appear very daunting.

On the flipside, if one breaks down the services into small groups and focuses on the services that are needed, it’s less daunting.

Overall, this project was still enjoyable. Working with AWS specifically, it was learnt that one should look at AWS and still if a service is available to one that is needed. An example of this was with TensorFlow and AWS Rekognition. The developer focused extensively on TensorFlow as they worked with it in the past and should have checked AWS for a similar service first. A lot of time could have been saved by switching to AWS Rekognition a lot sooner.

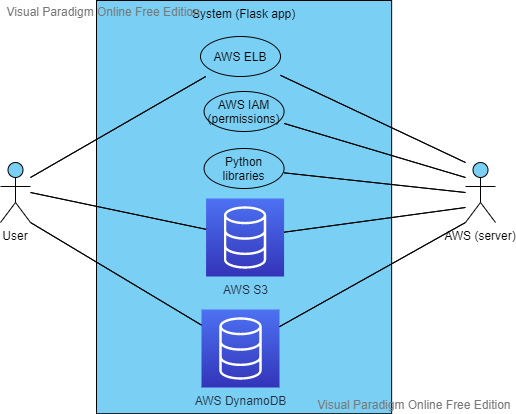
AWS Educate is closing on 31 December 2021, it was felt that as the service was at the end of its life, that some permissions were a bit skewed, that one set of permission wouldn’t work resulting in errors that could not be fixed. It was felt that the experience of creating a webapp was a bit hindered due to issues with AWS Educate.

The things that were liked were the used of the Boto3 library. This made connecting the app to the web services so much easier than having to use separate libraries for each AWS service. The use of the Cloud9 IDE was also enjoyable. The ease of setting up a Cloud9 IDE was simplistic and was easy to use.

# References

|  |  |
| --- | --- |
| [1] | J. Tatú, “Automatic Retinopathy Detection Using Digital Image Processing via a Smart Device,” National College of Ireland, Dublin, 2021. |
| [2] | R. Nieva, “Forget rainbow vomit. Google Lens is AR you can actually use,” CNET, 18 May 2017. [Online]. Available: https://www.cnet.com/tech/mobile/google-lens-is-augmented-ar-you-can-actually-use-snapchat-facebook-instagram/. [Accessed 1 December 2021]. |
| [3] | A. Villas-Boas, “Google's new Lens product uses your phone's camera to do clever tricks, like connecting your phone to a WiFi network,” Business Insider, 17 May 2017. [Online]. Available: https://www.businessinsider.com/googles-lens-feature-can-connect-your-phone-to-wifi-using-your-camera-2017-5?op=1&r=US&IR=T. [Accessed 1 December 2021]. |
| [4] | S. Campbell, “Flask vs Django: What’s the Difference Between Flask & Django?,” Guru99, 1 November 2021. [Online]. Available: https://www.guru99.com/flask-vs-django.html. [Accessed 1 December 2021]. |
| [5] | Amazon Web Services, “What is Amazon S3,” Amazon Web Services, 1 December 2021. [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html#CoreConcepts. [Accessed 1 December 2021]. |
| [6] | DynamoDB Guide, “What is DynamoDB?,” DynamoDB Guide, 15 December 2018. [Online]. Available: https://www.dynamodbguide.com/what-is-dynamo-db/. [Accessed 15 December 2021]. |
| [7] | S. Bhatt, “7 Reasons to Choose AWS as Your Cloud Platform,” DZone, 31 July 2021. [Online]. Available: https://dzone.com/articles/7-reasons-to-choose-aws-as-your-cloud-platform. [Accessed 13 December 2021]. |
| [8] | Amazon, “AWS Elastic Beanstalk,” Amazon, 13 December 2021. [Online]. Available: https://aws.amazon.com/elasticbeanstalk/. [Accessed 13 December 2021]. |
| [9] | TechVidvan, “Top 8 Exciting Features of TensorFlow to Know,” TechVidvan, 19 December 2021. [Online]. Available: https://techvidvan.com/tutorials/tensorflow-features/. [Accessed 19 December 2021]. |
| [10] | Amazon, “The Facts on Facial Recognition with Artificial Intelligence,” Amazon, 19 December 2021. [Online]. Available: https://aws.amazon.com/rekognition/the-facts-on-facial-recognition-with-artificial-intelligence/. [Accessed 19 December 2021]. |
| [11] | AWS Developer Guide, “Detecting faces in an image,” Amazon, 19 December 2021. [Online]. Available: https://docs.aws.amazon.com/rekognition/latest/dg/faces-detect-images.html. [Accessed 19 December 2021]. |
| [12] | AWS Developer Guide, “Moderating content,” Amazon, 19 December 2021. [Online]. Available: https://docs.aws.amazon.com/rekognition/latest/dg/moderation.html. [Accessed 19 December 2021]. |
| [13] | J. Rigden, “Tweepy: a Python Library for the Twitter API,” Medium, 22 January 2018. [Online]. Available: https://medium.com/@jasonrigden/tweept-a-python-library-for-the-twitter-api-9d0537dcebd4. [Accessed 19 December 2021]. |
| [14] | N. Samuel, “Amazon S3 vs DynamoDB: 5 Critical Differences,” Hevo Data, 8 November 2021. [Online]. Available: https://hevodata.com/learn/amazon-s3-vs-dynamodb/#factors. [Accessed 13 December 2021]. |
| [15] | Amazon, “Amazon Simple Queue Service,” Amazon, 13 December 2021. [Online]. Available: https://aws.amazon.com/sqs. [Accessed 13 December 2021]. |
| [16] | Amazon, “Amazon Simple Notification Service,” Amazon, 13 December 2021. [Online]. Available: https://aws.amazon.com/sns/. [Accessed 13 December 2021]. |
| [17] | A. S. Gillis, “Datadog,” SearchITOperations, 13 December 2021. [Online]. Available: https://searchitoperations.techtarget.com/definition/Datadog. [Accessed 13 December 2021]. |
| [18] | AWS Developer Guide, “Boto3 documentation,” Amazon, 19 December 2021. [Online]. Available: https://boto3.amazonaws.com/v1/documentation/api/latest/index.html. [Accessed 19 December 2021]. |
| [19] | A. Ronacher, “Flask,” Pocoo, 19 December 2013. [Online]. Available: https://web.archive.org/web/20171117015927/http://flask.pocoo.org/docs/0.10/foreword. [Accessed 19 December 2021]. |

# Appendix



This is an image of the Use Case and how a user interacts with the System.

Diagram

Description automatically generated

System Architecture design.

A screenshot of a computer

Description automatically generated

Code showing the creation of the S3 Bucket and running the server in AWS Cloud9

Graphical user interface, text

Description automatically generated

Setting up and checking app routes (pages) are working in the webapp.

A screenshot of a computer

Description automatically generated with medium confidence

Setting up AWS DynamoDB and testing with a batch-write file that adds 3 entries.

A screenshot of a computer

Description automatically generated with medium confidence

Deploying the app onto AWS Elastic Beanstalk

A screenshot of a computer

Description automatically generated

Checking logs of deployed environment for errors.

A screenshot of a computer

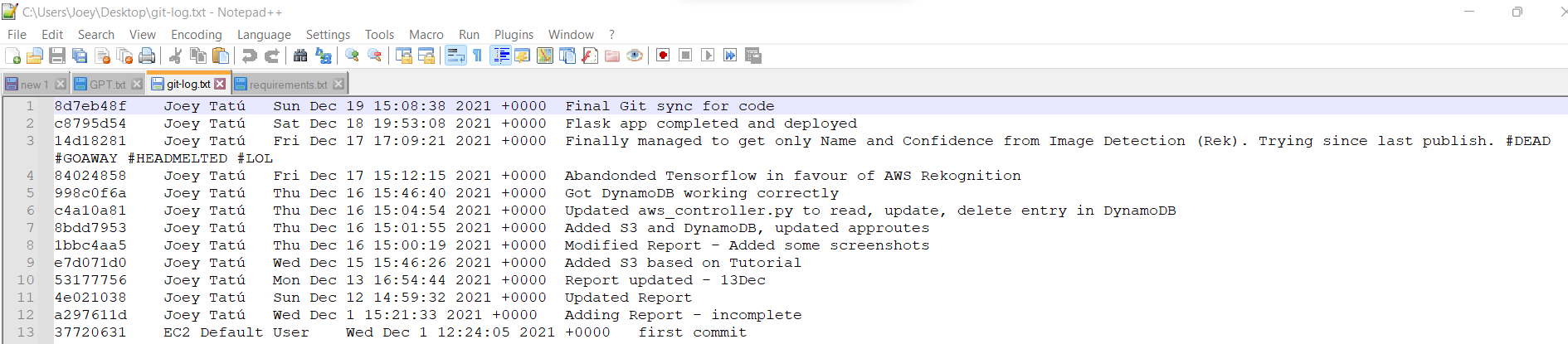
Description automatically generated

**Some** of the attempts of launching the environment on AWS Elastic Beanstalk

A screenshot of a computer

Description automatically generated

Edited inbound security rules to allow the port 8080 connection for the flask app.



Git log after completing code. Subsequent pushes for this report are not included.

Lecturer: Full Git log submitted on Moodle with code.