ALFRS – Cloud Solution

A Thesis Submitted in

(Partial) Fulfillment of the

Requirements for the Degree of

Bachelor of Science

in Information & Communication Technology

at

Bahrain Polytechnic

January 2021

Title

Copyright

Declaration

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

Signature: Name: Date:

Approval Signatures

APPROVED FOR THE ICT PROGRAMME

_____

Thesis Supervisor Date:

_____

Technical Writing Tutor Date:

Abstract

Depending on the purpose of the established events or exhibitions, the total number of attendees varies. Especially in Bahrain, as it became a worldwide tourist attraction. Tourists or residents tend to enjoy their holidays by seeking events, festivals, or exhibitions. These events attract different kinds of people regardless of their gender, age group, and income. This thesis aims to solve the problem that event organizers face when manually count and segment visitors and provide an automated solution that presents visitors' final statistics when an event concludes. The used technique to build the solution was to gather the client's requirements by examining their current system. Upon the outcomes, the causes of their problem were diagnosed by cause-and-effect methodology. The solution was designed to solve the issues and fulfil client requirements. Furthermore, the implementation included Artificial Intelligence services that detect and recognize faces and a cloud dashboard that presents live statistics to the client. This product supports the hypothesis that the automated system enables event organizers to count and segment visitors accurately while cutting operational costs and discharging manpower to perform other vital tasks. Thus, this product is beneficial for event organizers in generating statistics on visitors and staff management.

Keywords: Face Detection, AWS, Emerging Technologies, Face Recognition, Computing Technologies, Emotion Detection

Acknowledgements

First and foremost, I am incredibly grateful to my supervisor, Dr. Philippe Pringuet, for his invaluable advice and guidance throughout the project development. Your expertise and support motivated me to push myself even further than my limits, especially for your insightful feedback that played a vital role in my journey.

Secondly, I would like to thank my colleagues from my internship challenge at Bahrain AWS for their fantastic teamwork and enthusiasm toward being participators in a remarkable experience. Also, I would like to acknowledge their encouragement and assistance in challenging situations.

Thirdly, I would like to thank Tamkeen and Bahrain AWS for running an unforgettable, rich experience as well as providing the necessary tools and technology needed for accomplishing the project. Also, I would like to thank you for your guidance and recommendations to march in the right direction and ensuring the achievement of my objectives.

Fourthly, I need to thank Bahrain Polytechnic ICT Faculty tutors, Mr. Cyril Anthoni, Dr. Christos Gatzoulis and Dr. Alan Oxley, for their remarkable support in critical situations that I have been through in this journey. Also, I recognize your gratitude in building and preparing my intellectuality for the industrial field.

Lastly and most importantly, I would like to thank my parents and family for always being supportive when things got rough. Also, I would faithfully thank you for your continuous discussions, interests on my progress as well as being a way to relieve my stress that I have been under for in these tough circumstances.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Definition
AWS	**A**mazon **W**eb **S**ervices
AI	**A**rtificial **I**ntelligence
BTEA	**B**ahrain **T**ourism & **E**xhibition **A**uthority
APP	**App**lication
SNS	**S**imple **N**otification **S**ervice
API	**A**pplication **P**rogramming **I**nterface
SDK	**S**oftware **D**evelopment **K**it
IAM	**I**dentity and **A**ccess **M**anagement
SQL	**S**tructured **Q**uery **L**anguage
NoSQL	**N**on - **S**tructured **Q**uery **L**anguage
S3	**S**imple **S**torage **S**ervice
SES	**S**imple **E**mail **S**ervice
SNS	**S**imple **N**otification **S**ervice

Introduction

I. Project Rationale

Event organizers find it difficult to keep track of the number of visitors and segment them simultaneously. Weekend or holiday events are the toughest to process due to the vast number of attendees. Counting and segmenting visitors is a complicated procedure that causes unpredictable effects. In some scenarios, the number of visitors can be assumed or predicted before launching the event. Thus, it eases the preparation for event organizers (such as assigning staff and distributing equipment). However, if the number of visitors exceeds the expectations, this will cause a shortage of staff and equipment. In addition, this may result in inaccurate statistics that may mislead the company or the public. It is worth to mention if the number of visitors is less than the assumed number by margins, these will cause several issues. In essence, an increase in unneeded staff for counting and segmenting operation, loss from overly bought equipment.

II. Project Objectives

Project's primary objectives are to replace the manual counting visitor system with an automated approach that counts event visitors, accurately classify them based on gender and age range as well as generate real-time statistic reports for client review. The following list will detail the technical and business objectives of the new system.

- *Technical Objectives*
  - Deploy an automated AI cloud-based solution that process and host all system functionalities in the cloud.
  - Detect and recognize faces to improve the accuracy of segmenting and counting the redundant and unique visitors of BTEA events.
  - Maintain a reliable product by securing data confidentiality, integrity and accuracy within the production environment.
  - Build a scalable product able to handle massive amounts of concurrent requests.
  - Build and host a web app in a cloud environment to render real-time statistic reports for the client.

- *Business Objectives*
  - Increase exhibition revenue by increasing the rate of event sponsors and booths' rental.
  - Reduce the operation cost by eliminating payments for materials and equipment used by the previous system.
  - Free manpower to increase the productivity level in different event services.

III. Prior Work

The most relevant work to **ALFRS** product is the **Doorman** project by Sander van de Graaf. According to Graaf S. (2018), the DeepLens project helps detect and recognise

people within the office and sends Slack messages to the respective person when a face gets detected. The project work of Doorman differs from this paper's project. Doorman lacks the depth in implementation on extracting facial features of the detected faces and maintaining security measures on captured images. Furthermore, the other significant gap between the two works that ALFRS product operated in a real-time situation and displays data statistic on a Web App, whereas Doorman schedules the operation time and does not have an interface to display data. Thus, Doorman product count as insufficient to grade as it lacks some of ALFRS objectives.

IV. Hypothesis

This paper hypothesizes that the proposed automated AI cloud solution is more efficient and reliable than the current manual operation in various measures. The proposed solution will focus on easing the procedure with minimal manpower and delivering accurate real-time statistic records that the client can view. This solution will also be more secure and cost-efficient since the solution uses cloud services rather than the current system approach. Therefore, the proposed product's expected results are fast, secure, accurate, real-time and straightforward delivery towards the respective objectives.

V. Proposed Solution

The current manual system causes several issues. To overcome these issues, this paper proposes an automated system. This automated system solves the inaccurate statistics problem by detecting and recognizing faces using intelligence cameras; this process aims to differentiate between unique and redundant visitors. Also, this solution consists of a real-time dashboard that displays the statistics live to the client. Moreover, the dashboard will enable BTEA's staff to schedule new events. This functionality helps in triggering cameras and ensuring the automated system. After an event concludes, the statistics (not emotion statistics) will be gathered in a PDF document and sent to the event organizer. Furthermore, the dashboard enables BTEA staff to view previous event statistics along with their emotion statistics. The below table highlights the tasks that I am responsible for implementing.

Tasks
• Detect & Recognize Faces • Create user registration & authentication system for dashboard • Generate emotion statistics • Camera monitoring (Deeplens Heartbeat)

Table 1 My Tasks

VI. Description of the Report

The following sections in this paper will include a thorough discussion of the proposed solution. Each section will represent a particular topic to discuss, highlighting the approaches that were followed throughout the project. Starting with the Background section that will describe the necessary background information on some technical terms along with brief knowledge on utilized technologies. Second, comes the Requirement and Design section, this section involves highlighting requirement

methodologies that will be followed. Also, this section discusses the design approach that the proposed product will follow. Next comes the Implementation section; it describes the steps taken to build and implement the product. The Fourth, Testing section is the most critical section as it will consider all the accepted procedures that will ensure the product's efficiency and effectiveness. Lastly, the Discussion and Conclusion section will cover all the achieved objectives and their obstacles, along with any suggestion on future upgrades to this project. It will also include LESPI and a reflection on the gained experience.

Background

I. Related Theory

The number of attendees of an event is vital for event organizers as these numbers have potential benefits from social and economic perspectives. The number of visitors is the most critical factor that scales the event's popularity. The higher the number, the higher the event gets rated. There are a variety of visitor counting approaches. First is the manual operations (such as clickers) used for counting visitors. This approach is getting more complex, costly, and prone to human error due to the steady increase of attendees (Bek & Monari, 2016). Inaccurate counting of attendees results from several causes. One of the possible reasons is that event organizers tend to count visitors based on the number of sold tickets rather than the actual attendees. The same case goes for free-to-view events (Davies, Ramchandani, Coleman, 2010). Second possible cause, the ease of number manipulation by event organizers; they tend to inflate the number of visitors to often conclude with a higher event evaluation (Hara, Severt, Shapoval, 2016). Furthermore, the second approach is an automated technique. This paper will be discussing in-depth one of the automated approaches for counting and segmenting visitors.

Overall, narrowing the objectives towards one theory, this paper considers counting visitors as its primary concept. The following list presents the main used techniques to achieve with the finest solution to this theory.

- *Cloud Computing/Environment*

  The cloud environment is a versatile technology that can provide on-demand services at a lower price than full ownership cost. Services such as storage, network, servers, apps and many more are maintained and controlled by a cloud service provided in their respective datacenters (such as Google, AWS or Microsoft). The **Cloud Computing** term refers to deploying services on cloud service provider datacenters rather than deploying it on the premises of the organization. The cloud environment has several essential characteristics demanded by cloud clients: cost-efficient, time-friendly, high availability, flexible and scalable services (Jagirdar, Venkata, Reddy, Qyser, 2013). According to Shalala & Bokhara (2016), the cloud has multiple computing models such as Software as a Service (SaaS), Infrastructure as a Service (IaaS) and Platform as a Service (PaaS), each model has its criteria and driven aspect. Furthermore, with the evolution of technology, many other models came to the surface. A perfect example would be Function as a Service (FaaS). This model provides an event-driven environment that invokes a container that contains a code to run (Fernando, 2017). The proposed solution will utilize several of the mentioned models, as seen in the Project Technology section.

- *Face Detection & Recognition*

  Every human face has its extreme characteristics and dynamic structure. Recognizing faces can be quickly done by humans. In contrast, machines go through a range of activities in order to achieve this process. This machine process is called face detection and recognition (Hazim, Al-Dabbagh, Esam, 2016). Furthermore, the face recognition process determines faces in an image or a video by detecting facial

features such as eyes, nose, eyebrows and lips. Machines achieve this process through multiple algorithms and Machine Learning techniques based on AI technology. Facial detection has various application areas such as live-tracking people, entertainment or law enforcement (Itrat & Tabish, 2020). Furthermore, face recognition is dependent on the detection operation, as facial features are saved in a record and eventually represent a single individual. The newly detected face will be verified and identified based on the comparison with existing face records.

II. Project Technology

This is the most vital section to understand; the proposed solution is fully dependent on several technologies. These technologies (such as services, hardware and software) will be discussed thoroughly in the upcoming table.

Technology	Purpose
Cloud Technologies	
Amazon Web Services	During the implementation phase, this project used AWS for deploying and managing cloud instance/services. AWS is a cloud service provider owned by Amazon; is a robust cloud computing platform (Petters, 2020). AWS is the leading cloud platform in the **region**. It has more features and services than other cloud providers that provide low latency connection due to the short distance (AWS, 2020). On the other hand, Microsoft Azure would be the best alternative for AWS as it slightly reaches the same range of services and features. However, providing low latency connection to cloud resource could be vague compared to the regional dominator AWS (Butler, 2016).
 *Amazon DeepLens Camera*	Amazon DeepLens is an intelligent hardware camera capable of integrating with AWS service and performing complex operations. According to Fernandes J. (2017), DeepLens Cameras were designed for developers to run deep learning models with minimal challenging tasks. Therefore, this project utilizes this hardware to integrate with Amazon Rekognition for advance face-image analysis. There is no alternative technology to consider, as it is the only camera capable of executing codes and process machine learning models.
 *Amazon Lambda Function*	Amazon Lambda is one of the popular FaaS that provides computing and run-time environments for multiple languages without infrastructure management. Lambda functions have other astonishing benefits, such as auto-scaling instances in a massive number of requests, consistent performance in function execution and cost-friendly due to pricing per computing time (Hendrix, 2014). Alternatively, Amazon EC2 instance could be used as it provides a high-performance and secure computing environment. However, Lambda functions are more fit for this project in achieving a real-time batch processing due to their capabilities (Akiwatkar, 2017).
 *Amazon Rekognition*	Amazon Rekognition is a powerful service that can identify people, objects and text in videos and images. According to Mishra A. (2019), Amazon Rekognition can provide high accuracy in facial analysis and facial comparison. Therefore, utilizing this service will allow the opportunity to detect, analyze and compare faces to achieve project objectives. Rekognition service has several critic benefits such as high scalability during a vast number of requests, and ready deep

	learning models that do not require machine learning expertise. Alternatively, Microsoft Azure Face API would be a possible approach due to having similar potentials in facial analysis as AWS Rekognition API (Bobriakov, 2018). However, since the project implementation will be mainly with AWS as a cloud provider, it would be complicated to integrate other services from other providers.
 *Amazon DynamoDB*	Amazon DynamoDB is a NoSQL database that provides fast and consistent performance along with scalability feature, which is use full for semi-structured data. According to Rangel D. (2015), DynamoDB is a durable database built-in backup and restores, security and in-memory caching. It also supports a massive number of requests per second by more than 20 million requests. Moreover, AWS RDS can be an alternative for DyanamoDB, as it provides more elasticity and higher read rate. However, DynamoDB has more scalability and performance, which is what the project needs (Jayendrapatil, 2017).
 *Amazon S3 Bucket*	AWS provides a file system storage service that supports a variety of file formats. S3 Bucket (Simple Storage Service) is used for unstructured data, and it offers high scalability, availability, performance and security to the stored data objects. According to Rouse M. (2018), S3 Buckets has a versioning feature; this feature used to make versions to an object once a copy or deletion operation is performed. Furthermore, the reason to choose this technology instead of DynamoDB or RDS, due to the fact that S3 bucket has a high throughput that can deal with traffic requests, especially if the requests comes from different items.
 *Amazon Cognito*  *Amazon Simple Email Service*	Amazon Cognito is an advanced approach for authenticating and authorizing users; it stores and secures users' personal information and provides several other features such as Multi-Factor Authentication (MFA). According to Sosnowski M. (2020), Cognito avoids writing back-end service scripts while minimizing code on the frontend. Furthermore, Amazon Cognito grants other services such as Amazon Simple Email Service (SES) the permission to send emails and notifications on its behalf. According to Rosse H. (2015), AWS utilizes SES to send and receive emails as well as enable other services to use this feature. Both of the mention services will be used for verifying user accounts of the proposed system. An alternative for SES could be Amazon Simple Notification Services, it a highly reliable service that sends notifications to other services in AWS. However, the proposed solution tends to be more professional with Emails as it portrays to the client that the product is well established in the industry field (Atbasoglu, 2014).
	AWS delivers a scalable API for developers to publish, create and monitor called Amazon API Gateway. This service provides a secure connection with various API types (such as RESTful and WebSockets). Also, it counts as the fastest deployment services that integrate with others due to its serverless infrastructure (Janakieam, 2015). Furthermore, API gateway acts as the front face of the Lambda function in the front end. This service is useful to get or put data from the

Amazon API Gateway	cloud with minimal cost and time. Alternatively, API can be created using a domain; however, it will be justified in the upcoming sections why this approach is not possible.
Scripting Languages	
 *Python*	Python is a programming language that provides multiple functionalities for multiple paradigms. Its main advantage is the ease of implementation for developers. Moreover, it focuses on the quality of the programming code instead of the quantity. According to Mindfire Solutions (2017), Python is compatible with a variety of systems and platforms and has a robust of libraries that add more functionality to the python program. Furthermore, this language still has its momentum in growing over the years, indicating that developers' vast support is available (Paul, 2020). JAVA could have been chosen, as it has the same functionality with more expertise in the programming field. However, Python is chosen due to its simplicity and high functionality with a shortcode.
 *JavaScript*	Java-Script (JS) is a scripting language that is mostly used by websites and interfaces. It extends the web app by providing more functionalities in aspects of forms, buttons or animation. JS provide an excellent running speed as it offloads the execution weight on the client's device (FutureHosting, 2019). Furthermore, JS is easy to use due to its programming syntax as well as it is easy to debug and monitor using the browser console (Subhajit, 2018). It opens a wide range of opportunities for other scripting languages such as Node.JS and JQuery.
Design (Diagram, Prototype, Web-App)	
 *Creately*	During the design phase, this project utilizes Creately Software to design different kinds of diagrams related to the proposed project. Creately provides users with high assessability in creating and storing diagrams into the cloud, as it can be accessed anywhere and anytime with internet service. Creately supports a wide range of diagrams and icons used in different fields as it also has its icon libraries (Walsk, 2011). Furthermore, this tool provides a diligent and simple designing material that enables asynchronous real-time collaboration for designing. Alternatively, CloudCraft could be used as it is more expertized in AWS icons. However, CloudCraft has limited drawing space for icons. Thus, it restricts and impacts the overall look for a huge diagram (CloudCraft, 2015).
 *Balsamiq*	Balsamiq is a fantastic software tool used to design mockups and prototypes for web applications. It offers the fundamental design for a website as it provides a variety of icons, patterns and templates, which ease the design phase of a web app. This tool is very beneficial as it is utilized to present and test designs before agreement on the final design. According to Balsamiq, this software is the fastest tool in the industry focused on building a low-fi prototype. Marvel could be an excellent choice for an alternative, as it has the same range of functionality as Balsamiq, but it lacks the simplicity of use (Group, 2017).
	Hyper Markup Language (HTML) and Cascade Style Sheet (CSS) are the two primary elements utilized to build a website. While each component used to create interfaces, each serves a different purpose. HTML is used to display the web page component, whereas CSS is used to show how these components will look like on the web page (Sweeney, 2019). There are no alternatives for these factors, as they are still building a web page.

HTML & CSS	
Other Technologies Used by Colleagues	
Web Socket API	This type of API enables two-way interactive communication between the server and the client browser. Thus, it is the most suitable technology for implementing a real-time scenario case.

III. Related Work & Litrature Review

This section will demonstrate the related research on counting visitors. Since this paper's objective involves counting the number of people and using face detection and recognition to keep track of individual uniqueness, the research will be narrowed down to specific topics. Thus, the upcoming section will be divided into two coherent categories.

- ***People Counting***

The number of visitors of an event tends to change continually, keeping track of the number demands an extraordinary effort. Traditional methods, such as hiring staff to do the job became inadequate within the years. Sensors, on the other hand, is slightly coming to be insufficient. According to Jayaram M. (2016), people gathering with a high-density causes a lack of accuracy in the resulted number if sensors were used. The proposed solution of Jayaram involves a low-resolution camera that process images instead of sensors. The result of Jayaram's system shows it is capable of providing a sufficient accurate headcount with a real-time image processing; by the utilization of a low-cost and resolution camera. Although Jayaram's solution is related to this thesis, it lacks the solution of identifying a visitor's redundancy; since his solution only uses face detection via neural network and avoids recognizing them. Another related work is done by Roquero and Petrushin (2007). They had a similar approach as Jayaram but used a different technique for image processing; which is pixel density in each rectangle of the image. In comparison to this thesis, they lack the recognition approach to keep track of people uniqueness. Moreover, both of the mentioned work had similar disadvantages, for instance, the camera position should be exact, and the brightness is a critical factor.

- ***Facial Recognition in Cloud Computing***

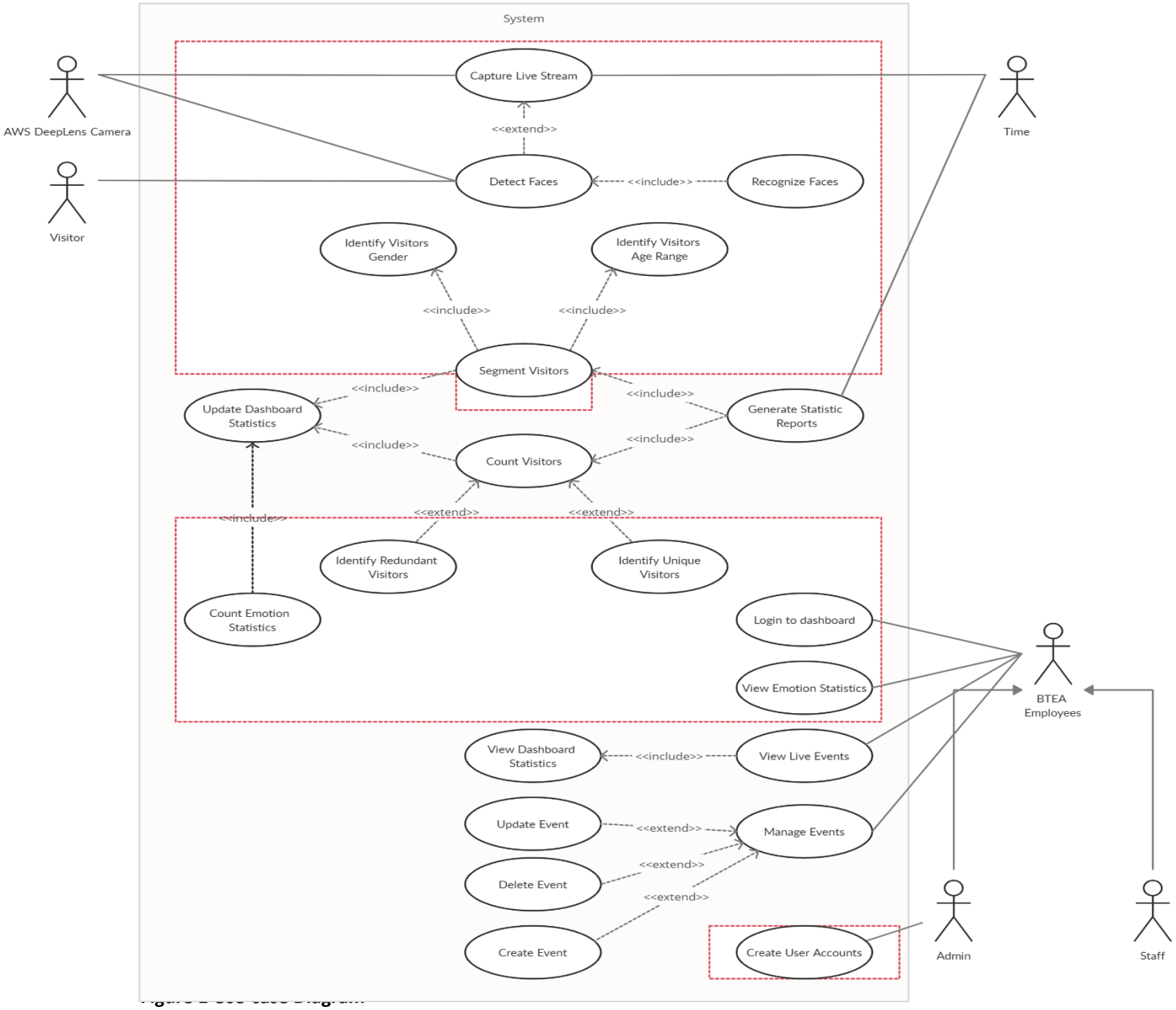
Cloud computing made face recognization a trending research topic over the last decade. The simplicity of deployment and low-cost are the main factors that attracted companies. The procedure of detecting and recognizing varies although the cloud companies provide their trained model to the public. According to Siregar, Syahputra & Rahmat (2018), processing an image for recognizing faces does not take that long. However, in the case of several faces in an image, operation time may increase rapidly. Thus, cloud computing is the ideal solution for a massive request as they provide scalable platforms. Siregar, Syahputra & Rahmat work used the Eigenface algorithm on cloud and utilized RESTful API to handle multiple requests while balancing the performance. The result of their work lacks several aspects in comparison to this paper. For instance, expose individual privacy

as images are sent to the cloud. Also, the Haarscade model that they are using needs trained data. In comparison to this paper, their solution had focused on creating a model for recognizing faces instead of using the trained models to simplify and ease the operation.

Methodology

Use-case Diagram

The following figure is a use-case diagram that represents the overall proposed system. According to Rouse M. (2020), use-case diagram summarizes details of the system and the users within the system by showing the interaction between them. This diagram is beneficial for gathering requirements. It highlights the prominent cases where the system functionality relies on and presents their major contributors or individuals relevant to the concern activity. Note, all task



surrounded by red dotted rectangles were completed by me while the rest of the system was implemented by my colleagues.

Prototype

This draft prototype is crucial as it identifies the design layout of the web app for developers. According to Mishra S. (2019), design prototypes demonstrate the future site's structure as its site maps and establishes interrelations between web pages. Therefore, using Balsamiq tool, a low-fidelity prototype was created. This prototype presents a complete idea of how the dashboard will look to the client and obtain their feedback. The design overview will also ease the designing and development process of the dashboard and reduce risk occurrence in the project compared to a project which prototyping was not carried out.

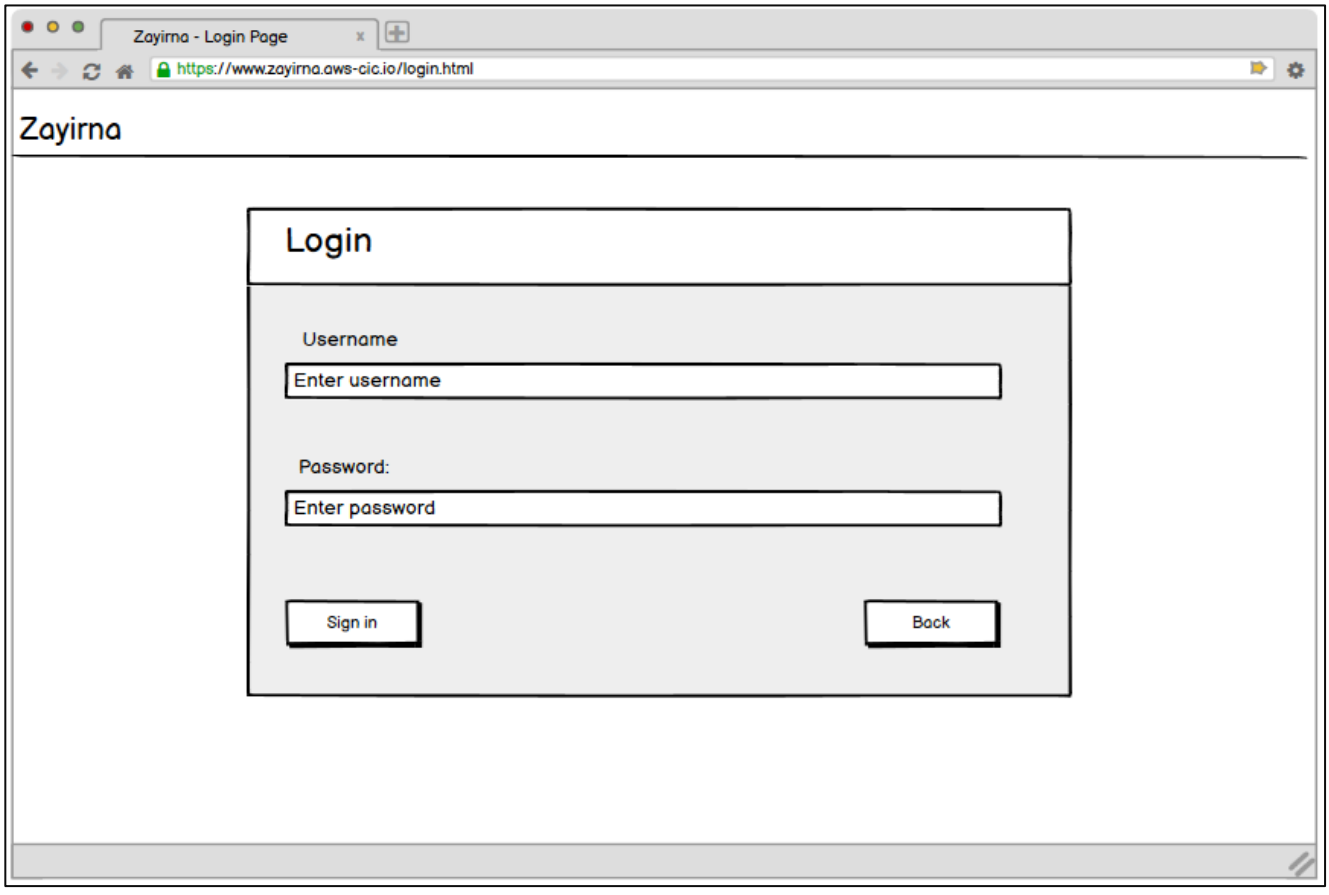


Figure 2 Prototype - Login Page

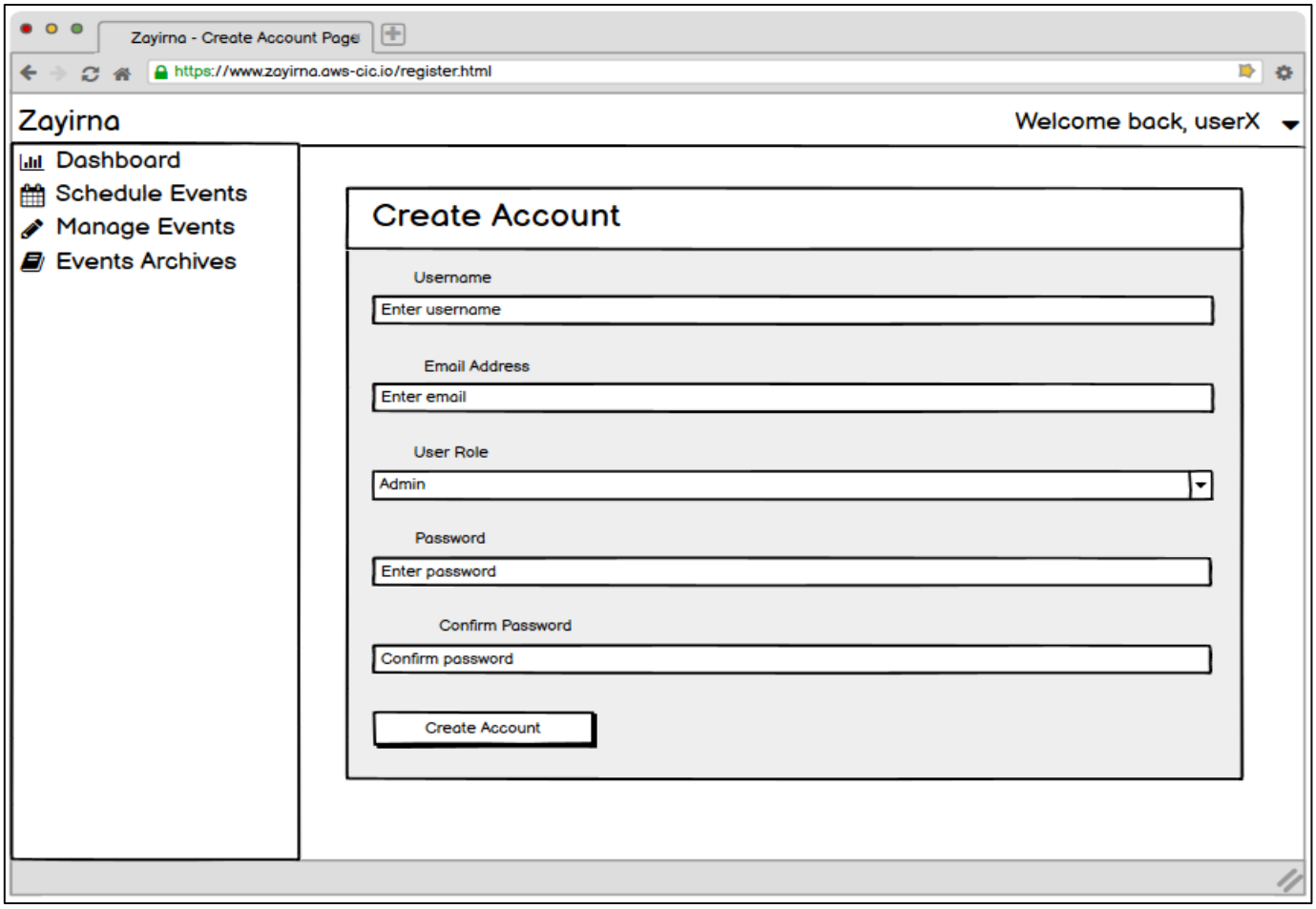


Figure 3 Prototype - Create Account Page

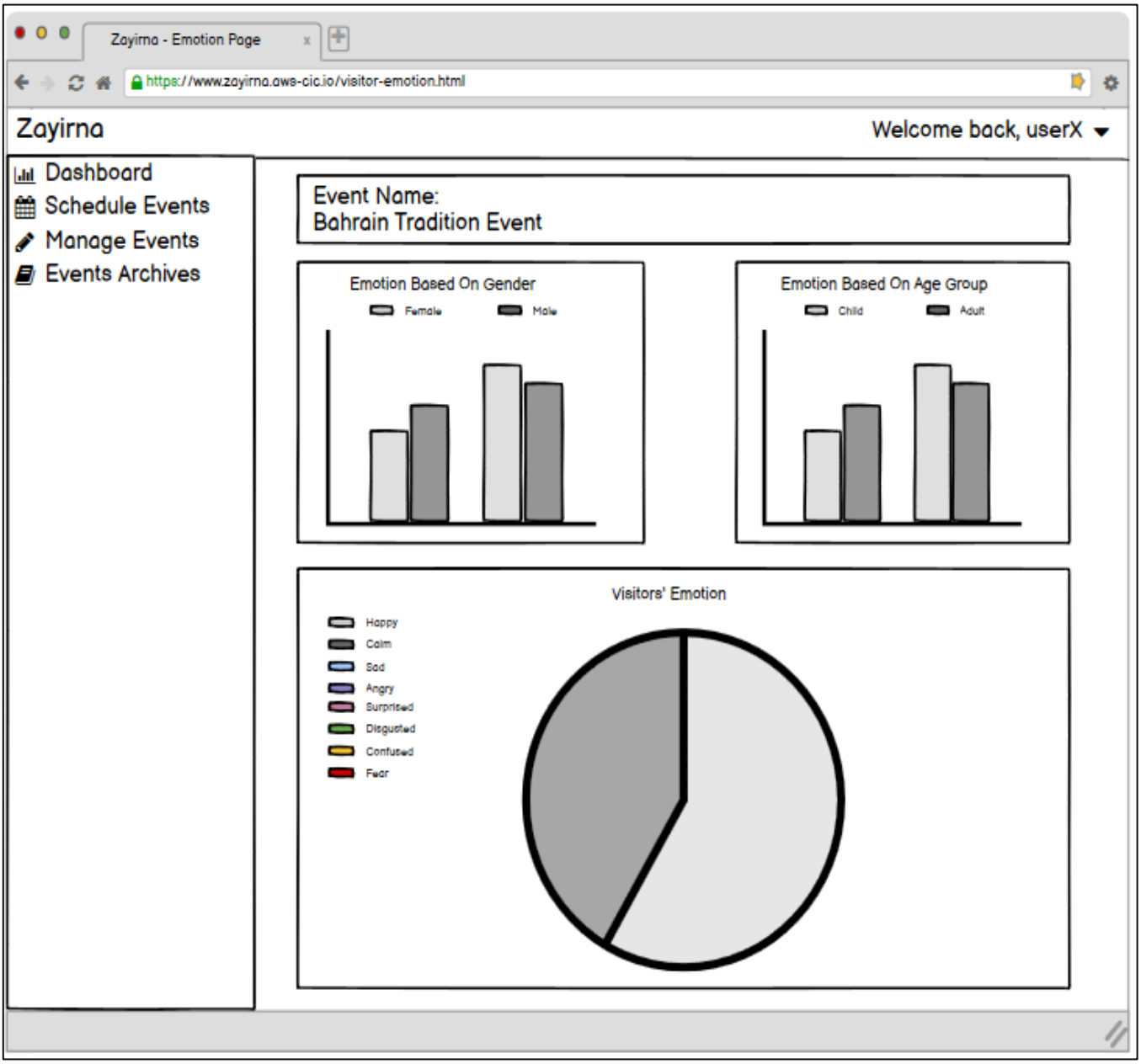


Figure 4 Prototype - Emotion Page

- ***Solution Design***

In brief, this section will describe all the design methodologies that took place within the project. This section is vital as it constructs the structure of the project before the implementation phase. This section will include the data structure, system architecture, deployment, activity and sequence diagrams. Furthermore, refer to the details design section in the appendix for additional diagrams on the authentication system, deeplens heartbeat and the emotion statistics generation system.

a) Data Structure

The project uses no schema database, also known as NoSQL database. Various reasons lead us to decide on NoSQL schema. The project generates a massive number of records from multiple sources. According to Fowler A. (2017), NoSQL handles a vast amount of data as it provides horizontal scaling and supports various data structures. Moreover, AWS provides DynamoDB services (NoSQL Schema) that stores data in a key-value format. The key is stored in an index structure that enables quick records identification. Thus, the indexed key is an ideal approach for fast data retrieval. Furthermore, since the project does not use SQL schema, it will not include an Entity Relation Diagram (ERD). However, the below figure is a perfect model for visualizing NoSQL table schema as it is built using NoSQL Workbench.

Primary key		Attributes								
Partition key: partitionKey	**Sort key: metadata**									
event-ID-0313	ev-013	eventName	startDate	startTime	endDate	endTime	organizerName	organizerEmail	organizerPhone	noDays
		Bahrain Tradition	2020-12-12	20:20:00	2020-12-12	21:20:00	Ali hasan	alihasn@gmail.com	31209857	0
	fd-re-face-dep3-pfowkpnkwfew42	faceID	age	gender	ageGroup	isRedundant	emotion	occurranceTime	deeplensName	
		fsrf2-gsgq4g-h578-af4-v3v3v-y7ej	23	Male	Adult	True	Happy	2020/12/12 20:20:22.432121	dep3	
	fd-uq-face-dep1-f98gdg9yey9fur90	faceID	age	gender	ageGroup	isRedundant	emotion	occurranceTime	deeplensName	
		4wvv4wc-gs8r6id-6-3xd4tra4-6v5ct5se-x4gw	20	Male	Adult	False	SAD	2020/12/12 20:20:20.432121	dep1	
	noOfAdults	aggregateValue								
		2								
	noOfChildren	aggregateValue								
		0								
	noOfFemales	aggregateValue								
		0								
	noOfMales	aggregateValue								
		2								
	noOfRedundant	aggregateValue								
		1								
	noOfUnique	aggregateValue								
		1								

Figure 5 DynamoDB - Events Table

It can be seen from the above image that the data are stored differently in comparison to SQL tables. For instance, the primary key is a combination of two keys (partition key and sort key); both keys generate a unique key when combined to accelerate the data retrieval. Furthermore, NoSQL workbench helps in visualizing data by providing facets functionality. Facets are predefined patterns (columns) used to differentiate between different categories. From the above figure, there are three facets. The first row is the Event Information Facets that includes organizer and event information. The second and third-row are Face Information Facets; they contain the extracted features from face detection and recognition. The last facets are the Aggregation Information that includes the aggregated values of visitors used in the real-time dashboard. It is worth to mention that facets are just for visualizing purposes as the real implementation will not look like the above image.

b) Architecture Diagram

Architecture diagrams are very useful for reflecting the internal state of the product. According to Guthrie G. (2020), software system architecture diagrams break down the system into layers that display how different components interact. Therefore, architecture diagrams are beneficial to understand the behaviour of the system component as well as it eases the communication across stakeholders.

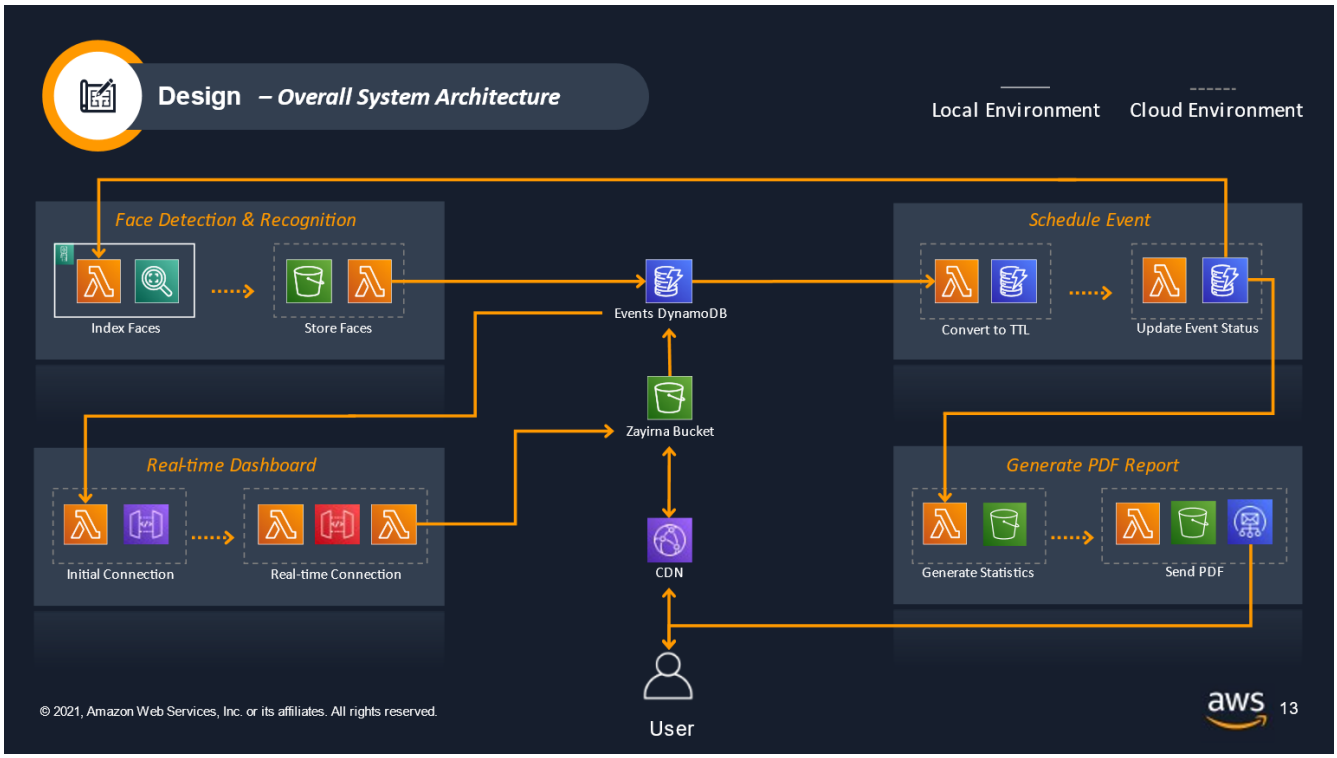


Figure 6 Architeicture Diagram - Overall System Diagram

The above figure presents the architecture diagram of the four significant functionalities in the final project. There are several additional functionalities of the product that makes the product more reliable such as archive events and

emotion statistics. Furthermore, the system captures, recognize faces using deeplens cameras and stores the faces records in the Events Table. Following this process, the records get extracted to generate real-time statistics to display them on the web application. The web application has several functionalities while the most important is to schedule a new event.
Users enter the event's time and date duration to specify when the cameras are needed to be triggered. Moreover, after the event has concluded, a pdf file containing the final statistics is being generated and sent to the event organizer.

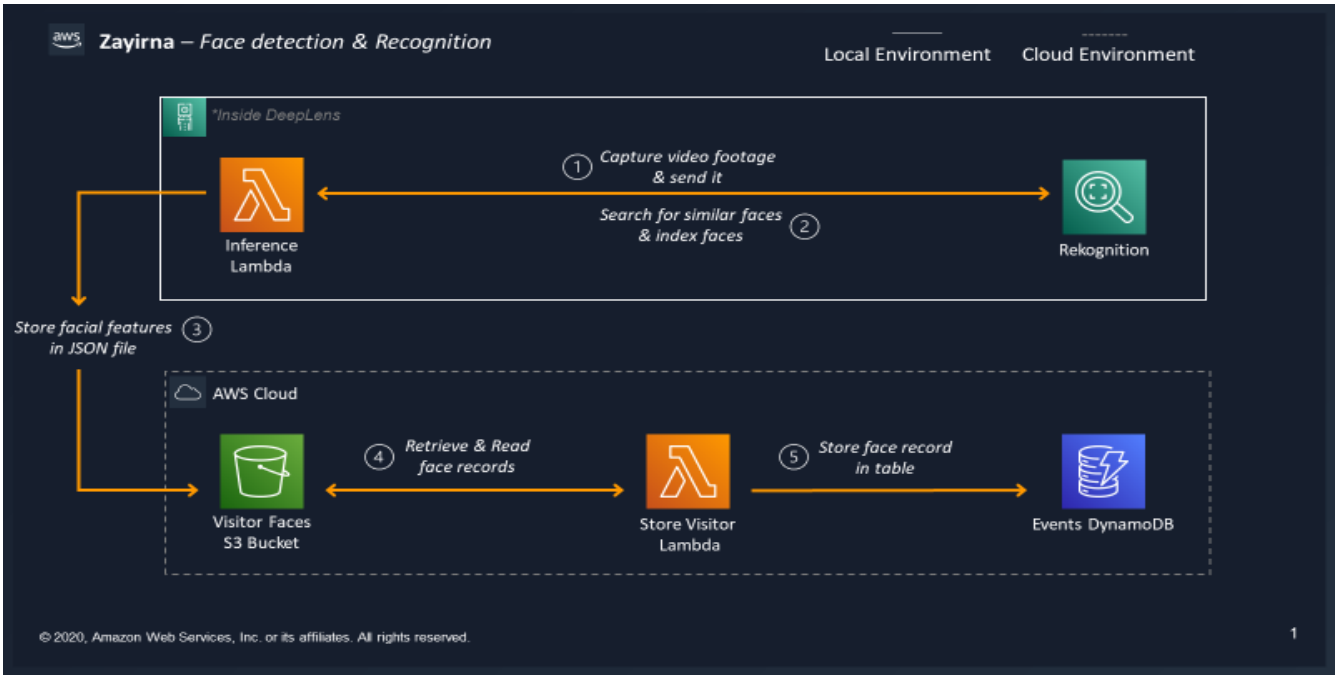


Figure 7 Architeicture Diagram - Face Detection & Recognition

It can be seen from the above diagram that there are two environments, the first environment is the Local environment which is the Amazon Deeplens Camera; the cameras' location is at the event entries or exists. The Deeplens process two main things: the first is to capture video footage, and the second is to detect and extract the visitors' facial features. AWS face detection model resides inside the Deeplens device, and this model performs face detection. Once a face is detected, the inference lambda will crop the image and send it to Rekognition. The Rekognition services will return a JSON format response contains detailed facial features. Moving to the second environment is the Cloud Environment. Once the Inference Lambda receives the response, it will store the data in a JSON file in the Visitor Faces S3 Bucket located in the cloud. While faces are fetched to the S3 Bucket, a lambda function will be triggered on each inserted JSON file. This lambda (Store Visitor Lambda) will open and read the file content and insert the extracted data into the Events DynamoDB Table.

c) Deployment Diagram

This diagram demonstrates the path of the execution environment between the system hardware and software. According to Visual Paradigm (2016), deployment diagram is a diagram that shows the configuration of run-time processing nodes and the system components. Therefore, this diagram will be beneficial for my client, as it will illustrate the structure of the run-time system and how the communication between hardware is established. The following figure shows the hardware division of the whole project. The first section from the left is the Deeplens Camera. This camera process the face detection and recognition operation and sends the data to the cloud. The cloud consists of several services; each server is dedicated to a specific job. For instance, the lambda server runs and compile codes while other servers such as S3 Bucket server are used for storage. After the cloud process, the records send the data over the Content Delivery Network (CDN) when a user requests to render data from the web app.

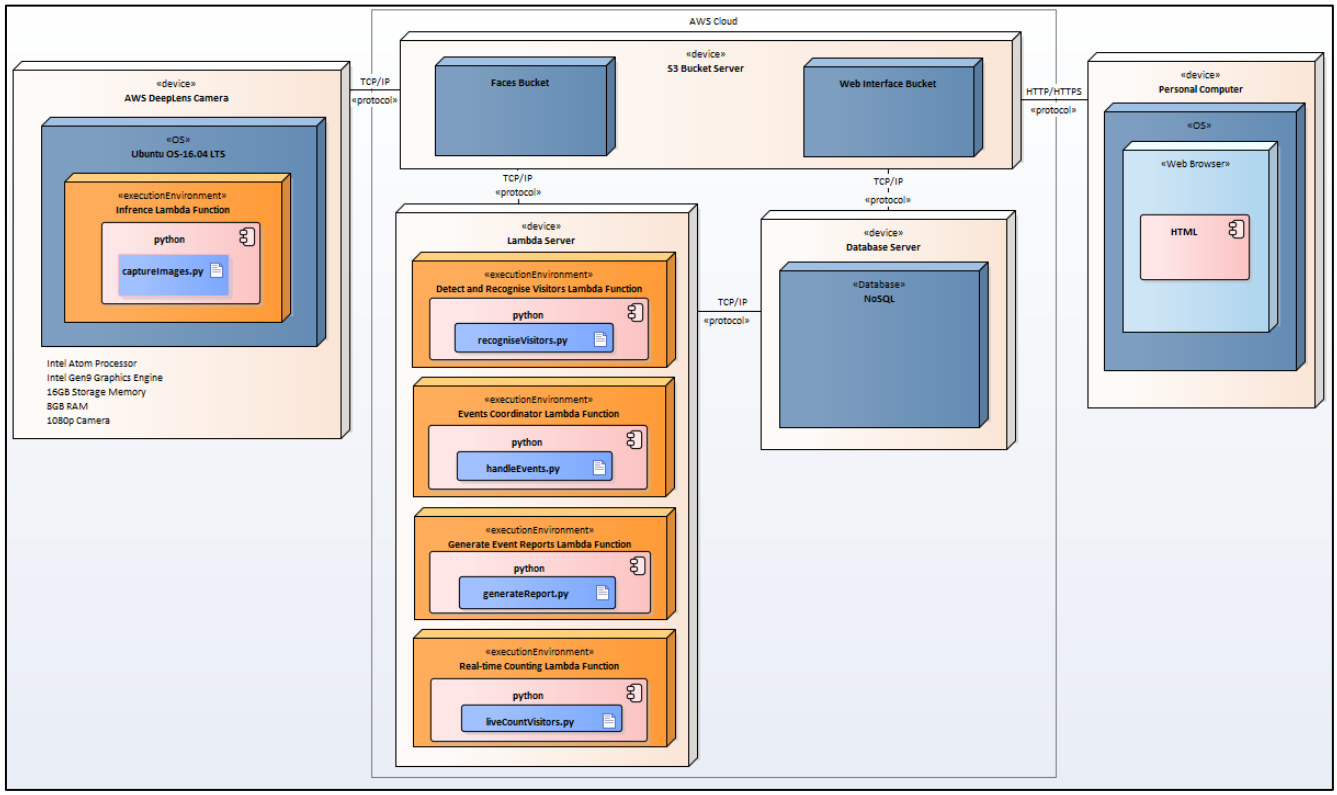


Figure 8 Deployment Diagram - Overall System

d) Activity Diagrams

Activity diagram illustrates the flow of action of a procedure. Usually, these diagrams represent an individual use-case, but sometimes they can contain multiple use-cases. These diagrams are perfect for modelling business processes as it demonstrates high-level actions that correlated to a specific operation in a system (Lynch, 2019).

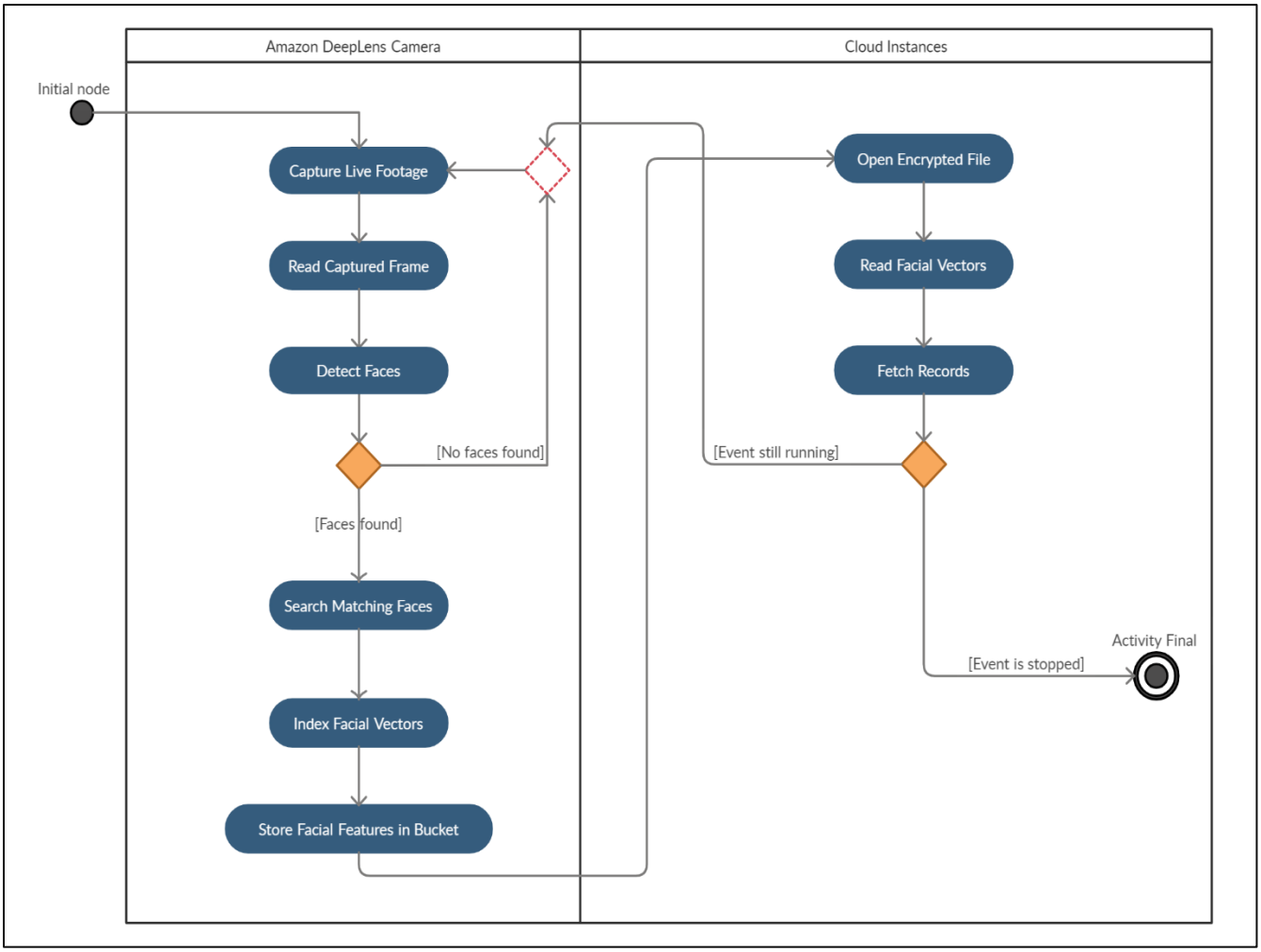


Figure 9 Activity Diagram - Face Detection & Recognition

The above diagram demonstrates the use-case of face detection and recognition. The use case initiates once there is a running event. The camera starts capturing footage and reads the frames for a face; it detects faces using the trained model located inside the Deeplens. Once a face gets detected, it searches for a similar face by comparing the input image with the available indexed records. This process extracts facial features into a feature vector and stores them in a JSON file in the S3 Bucket (Cloud). The file gets encrypted by the Server-Side Encryption (SSE-S3); only authorized services can open and decrypt the file. This approach performs as the project abide to maintain visitors' privacy. A process is waiting for new face records to read the facial features and store them in the data repository. The whole process is repeated until the event ends.

e) Sequence Diagram

Same as the activity diagram, this diagram counts as one of the Unified Modeling Language (UML) diagrams. Sequence diagram presents the interaction between objects in sequential order as it shows the lifeline of each object and the actions they performed. This diagram is beneficial to showcase scenarios in the system, and the lifetime of an object in the execution period. It also displays how data and messages are being passed between different objects (Larman, 2005).

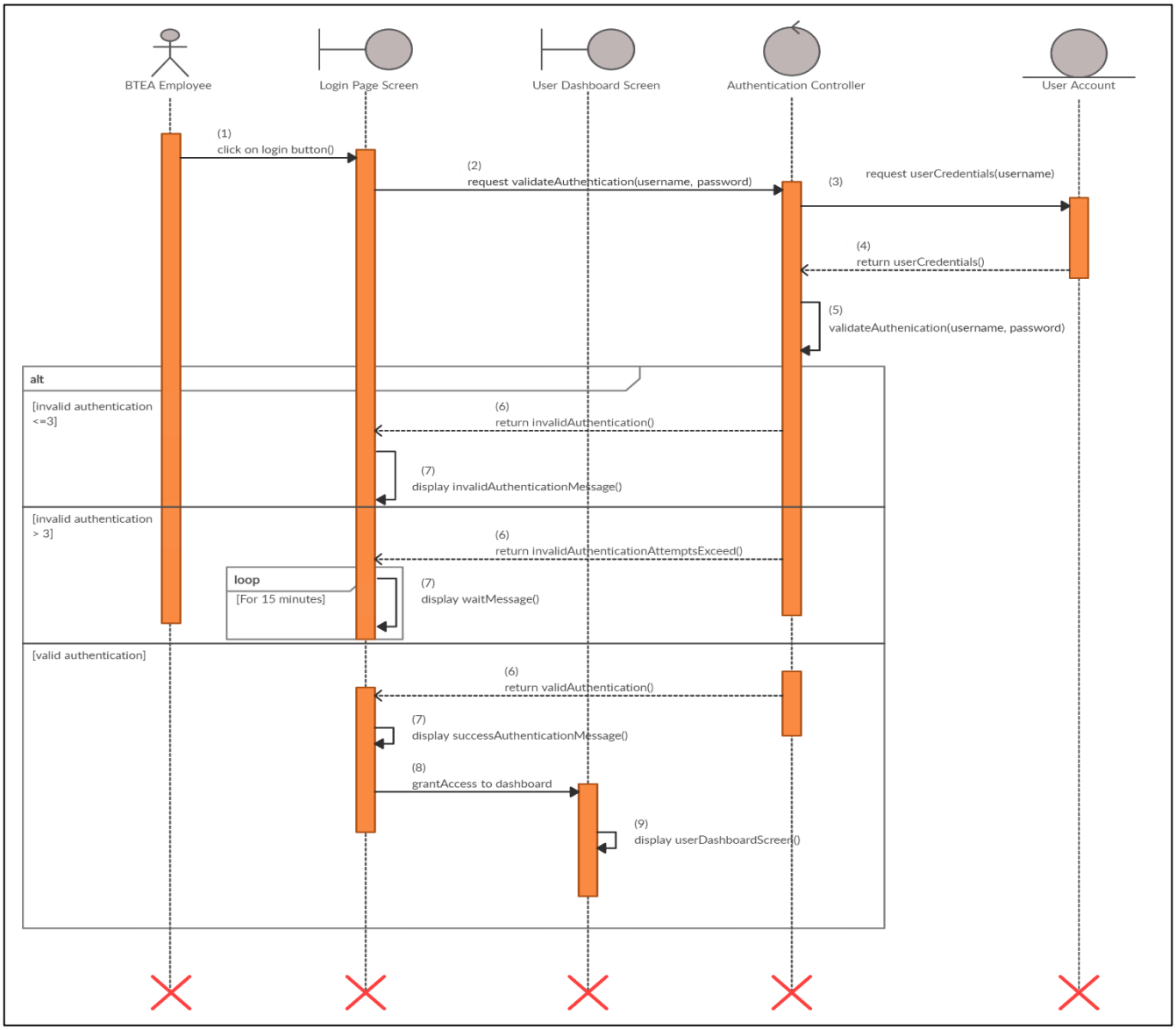


Figure 10 Sequence Diagram - User Authentication

The above diagram showcases the sequence actions when a user attempts to log in after entering their credentials. Once the user clicks the login button, a validation credentials request is passed to the authentication controller (Cognito

Service). The controller requests the data based on the entered username from the User Account; which is the stored records in the Cognito service. Once the handler validates the credentials, a respective message is displayed on the login page. In case of invalid attempts that exceed three times, the login page goes to a loop for 15 minutes showing the waiting period, and it resumes regularly once the duration is completed. In case of valid credentials, the user is granted access to the dashboard screen.

IV. Implementation

This section of this paper includes steps and procedures followed to deliver the final product. Detailed steps could be found in Appendix III section. Note that this paper does not have full implementation of Zayirna project. If required, please explore my colleagues' documents. This section contains the face detection and recognition process, user authentication and registration, emotion statistics generation, and the deeplens heartbeat. Note, all the following steps require logging in to an AWS IAM Account.

1. Face Detection & Recognition

This section includes registering the Deeplens device following by creating a project deployed into the device. This project will consist of a trained model and a Lambda function responsible for detecting and recognizing faces using the trained model. Once the lambda recognizes a face, it performs three actions. The first action is to search for a similar face and determine their redundancy. The second action is to index the detected face by extracting facial features using Amazon Rekognition service. The third action is to store the extracted face into an encrypted S3 Bucket.

Furthermore, another Lambda was created to retrieve the file stored in the S3 bucket and extract face features (such as gender, age, and emotion) and fetch the values into a record in DynamoDB.

- Step-By-Step Service Creation

The first step is to register the DeepLens Camera in the respective AWS IAM Account. In this step, AWS provides two versions of the Deeplens Camera, and as for this part, HW v1

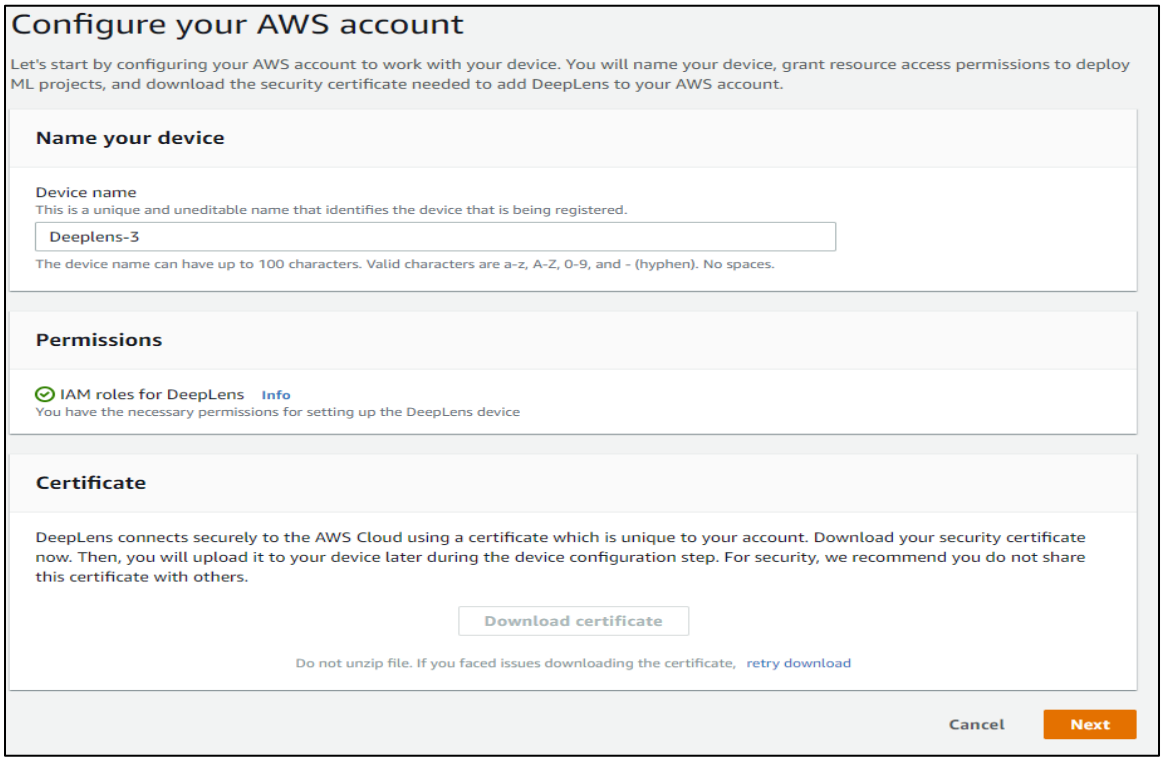


Figure 11 Implementation - Deeplens Configuration 1

(Hardware Version 1) was selected. After completing these steps, enter the deeplens name and downloaded the certificate as seen from the below image.

After completing the registration, the next step is to connect to the device and upload the previous step's downloaded certificate. Note, to connect to the camera; it needs to be in reset mode. This step can is accomplished by pressing the reset button for five seconds. The following figure presents the configuration screen of the Deeplens device.

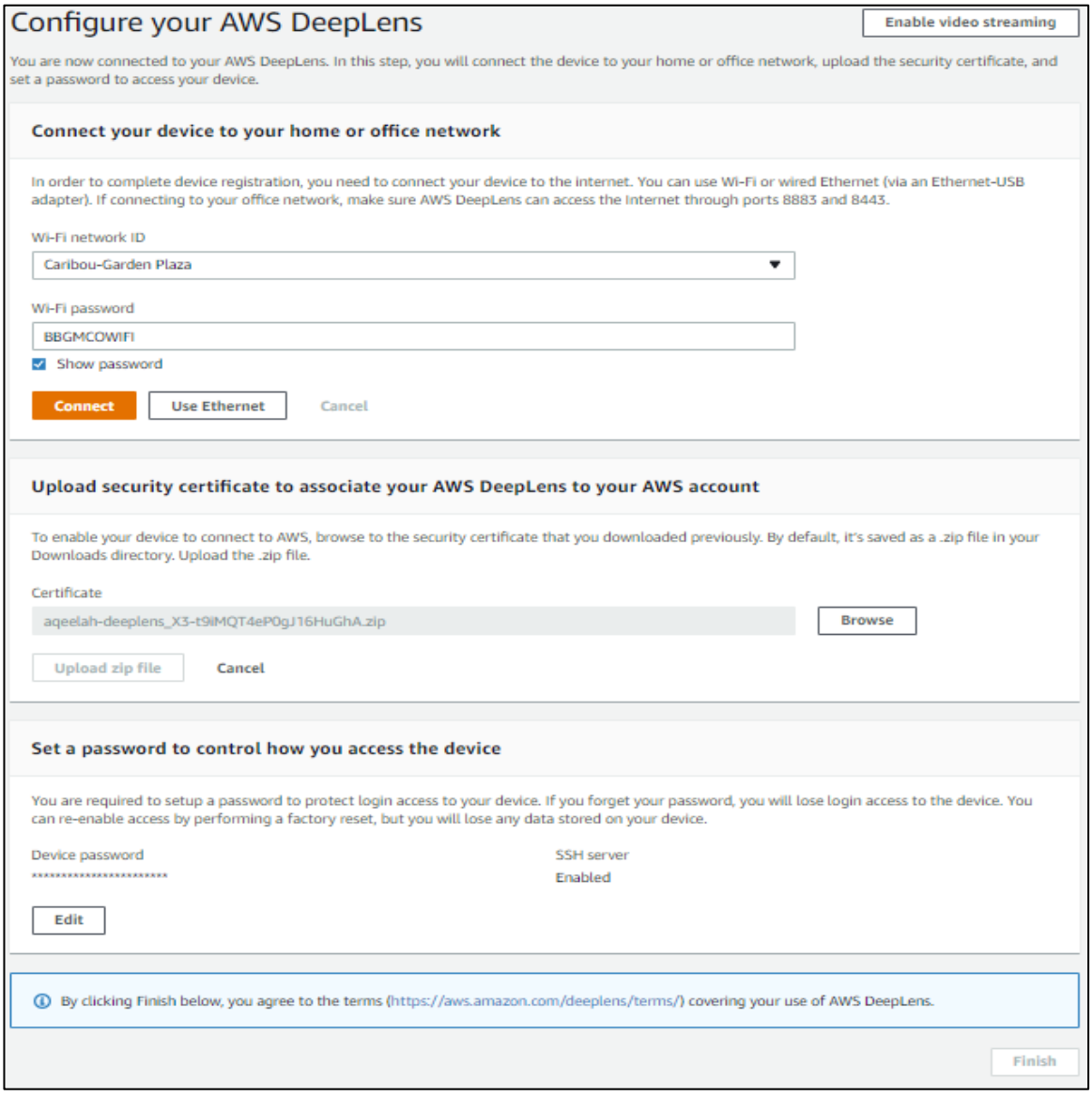


Figure 12 Implementation - Deeplens Configuration 2

After configuring the Deeplens, a Deeplens project was created. This project is responsible for the detection and the recognition of a face. It is worth to note that the outcome of completing a Deeplens project is a trained model called "deeplens-face-detection" and a lambda function called "deeplens-face-detection" that was deployed into the Deeplens device. Furthermore, to achieve this step, selected the "Projects" section from the AWS Deeplens Service's left side.

Following, created a project by clicking "Create new project" from the top right, then chose "Use a project template" for Project Type, "Face detection" for Project Templates and click Next.

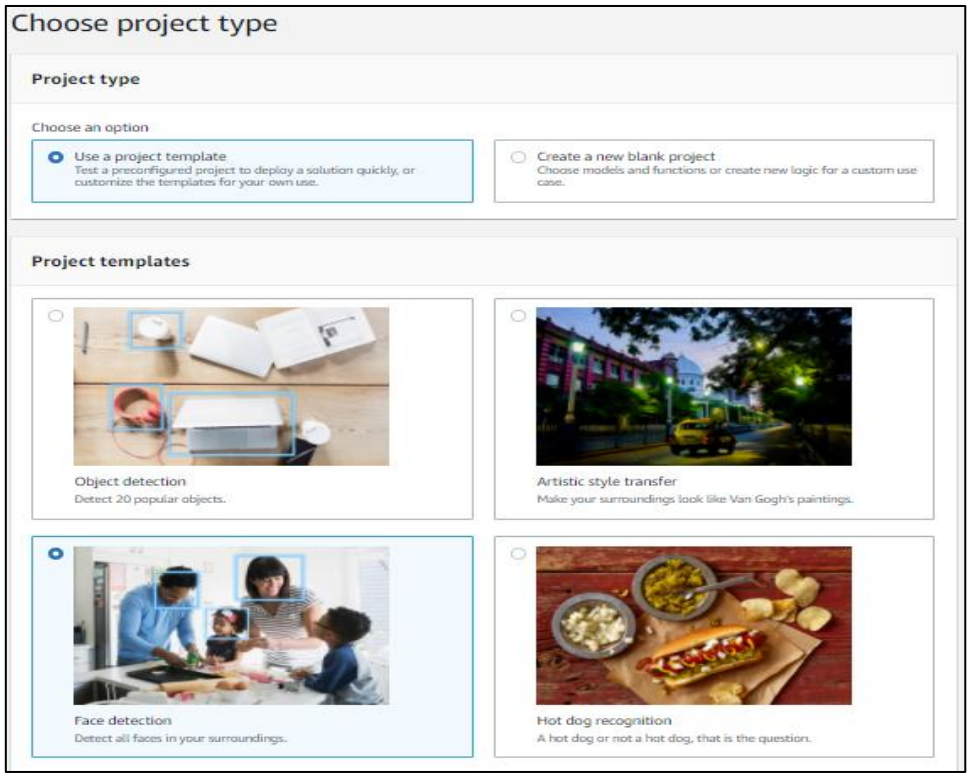


Figure 13 Implementation - Face Detection & Recognition (Deeplens Inference Lambda 1)

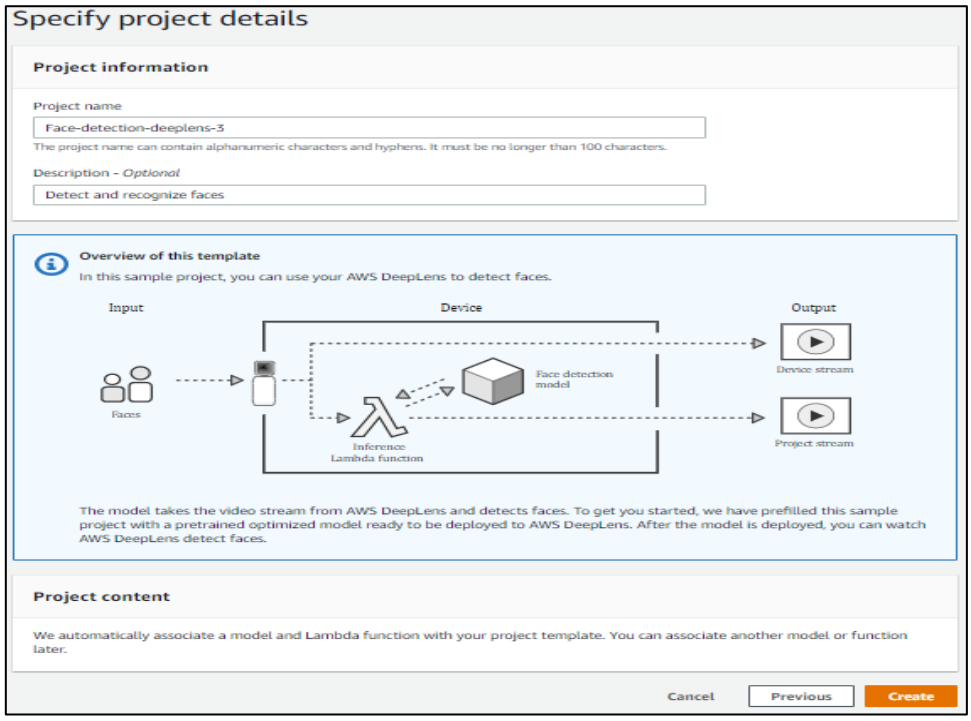


Figure 14 Implementation - Face Detection & Recognition (Deeplens Inference Lambda 2)

Furthermore, an encrypted S3 Bucket was created to store the outcome file of the Deeplens Camera. To achieve this step, search for Amazon S3 service and click "Create Bucket" from the right-hand side. Following, provide the bucket's name, blocked all public access and encrypted any new objects stored inside the bucket. The encryption approach is vital as it protects the critical data generated from the Deeplens device; the projects abides to protect the public privacy and ensure robust security to sensitive data.

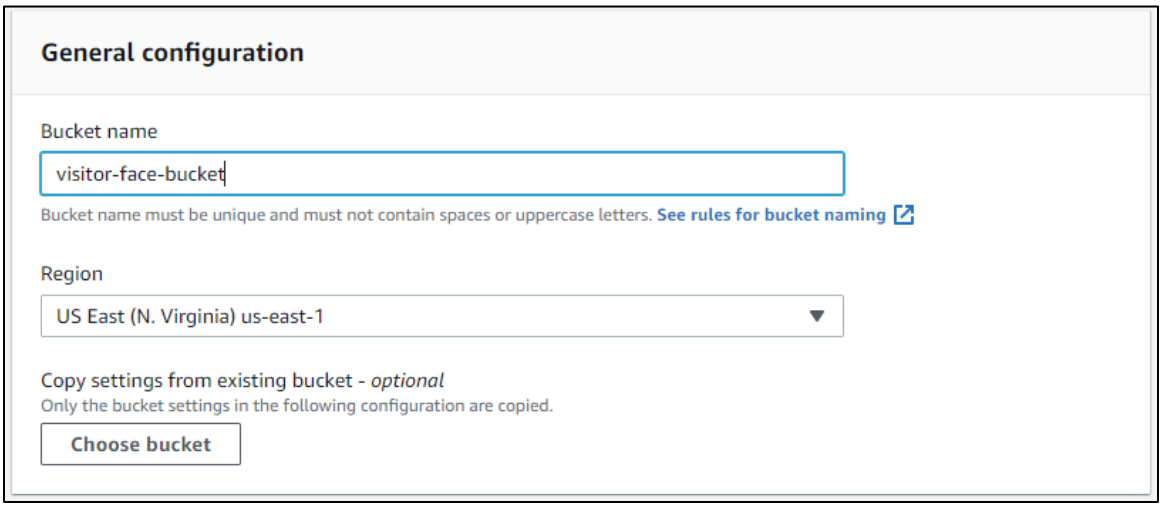


Figure 15 Implementation - Face Detection & Recognition (Bucket Name)

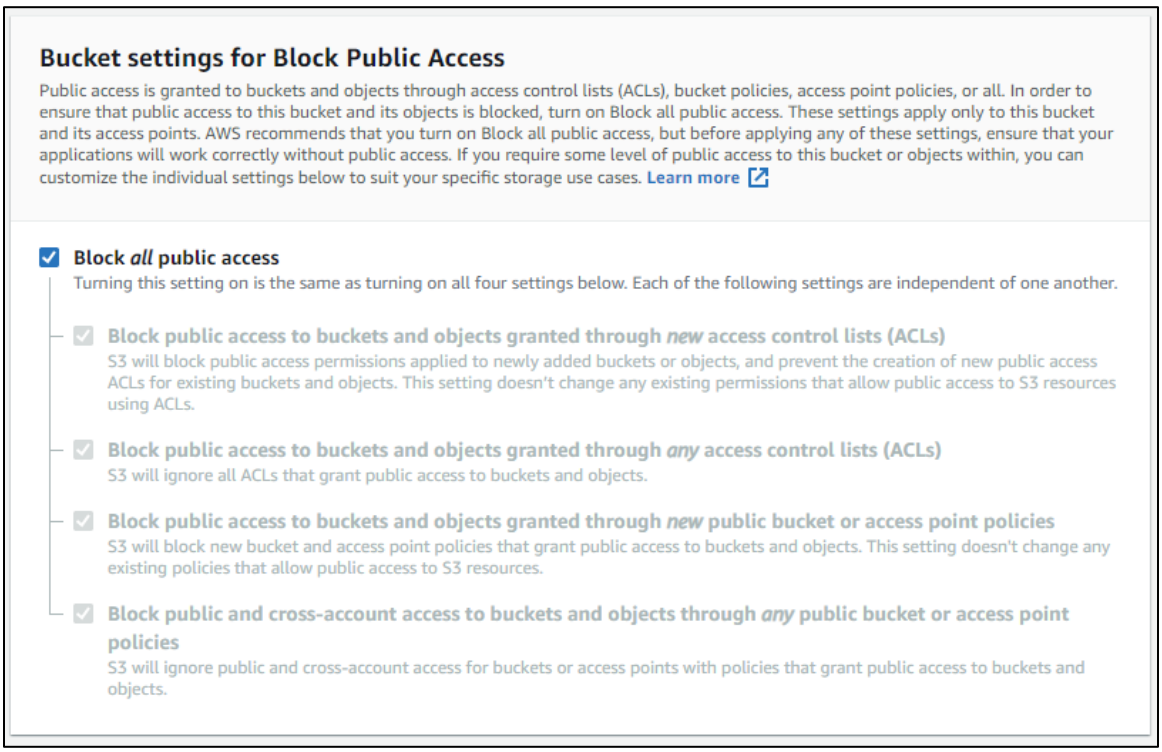


Figure 16 Implementation - Face Detection & Recognition (Bucket Access)

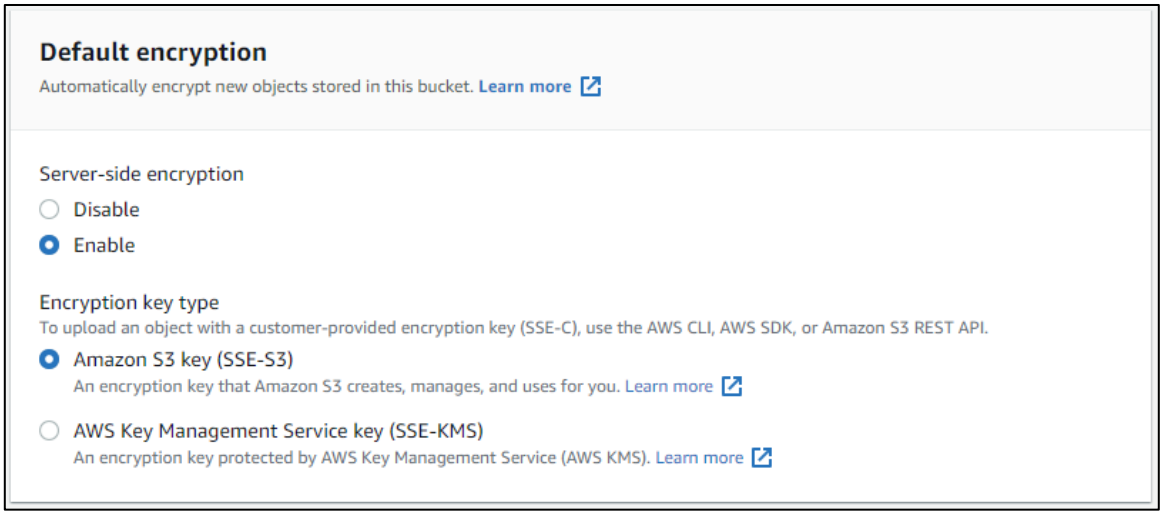


Figure 17 Implementation - Face Detection & Recognition (Bucket Encryption)

After creating the bucket, a lambda function called “Store-visitor-face” was created. It is responsible for retrieving the files from the bucket and insert face records in a DynamoDB table. The following figure presents the “Store-visitor-face” lambda function.

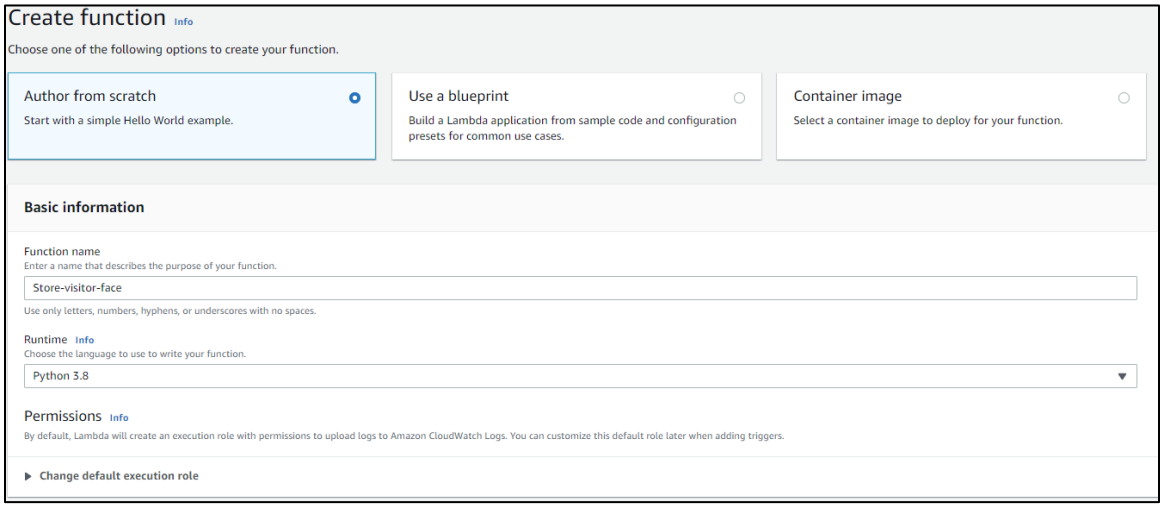


Figure 18 Implementation - Face Detection & Recognition (Store Visitor Face Lambda)

After creating the lambda, it has to configure a trigger which takes action once a new file has been inserted to the bucket. The following figure displays the S3 bucket previously created as a trigger to “Store-visitor-face” lambda function.

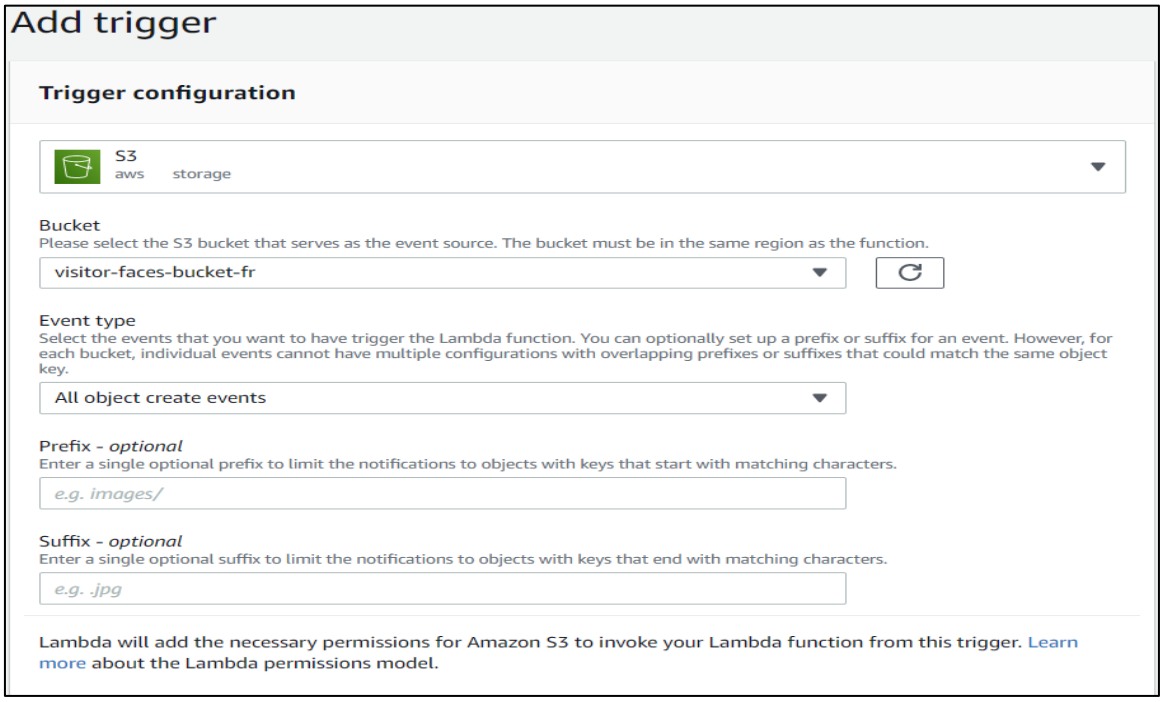


Figure 19 Implementation - Face Detection & Recognition (Store-Visitor-Data Trigger)

The inference and store-visitor-data lambda have several permissions that enable them to integrate with other services. Permissions are highlighted in detail in appendix section under the Detailed Imeplentaion. Furthermore, the appendix section also includes a detailed explanation of how inference and store-visitor-data lambda were programmed.

○ Alternative approach

The face detection and recognition part went through several approaches before reaching the final approach. For instance, a previous approach was to store the captured frame from the Deeplens in an S3 bucket. This approach was avoided later as the aim of on edge technology is not to let the image leave the Deeplens device. Besides, the previous approach causes the solution to be vulnerable to attacks, which makes people's privacy experience a critical situation. Therefore, the final approach processes the detection and recognition operation inside the Deeplens device itself, which means captured frames never leave the device. What does go is the facial features of the captured faces.

2. ***User Authentication & Authorization Via Amazon Cognito Service***

This section includes creating a user pool in Amazon Cognito service to store the information of BTEA users. This service requires specific attributes to create a new user account. Furthermore, an SDK is needed for establishing communication between the browser and the Cognito service. The browser utilizes the SDKs to log in or sign up procedures; it creates Cognito objects and calls specific functions that perform the actions. In signing a new user, Amazon SES service is used to send a genuine verification email to the new user.

- Step-By-Step Service Creation

The first step is to create a user pool in Congito that uses the username to sign in and sign up. Attributes such as email, user role and password are also needed to create an identity for a user. The below figure shows the option for users to sign in using the username.

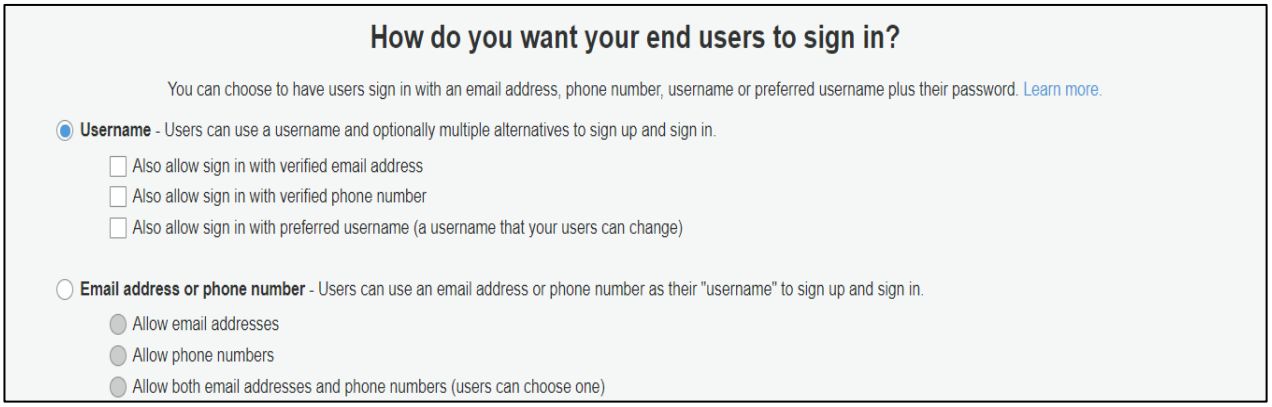


Figure 20 Implementation - User Authentication (Username)

The email attribute is selected to verify the new user by sending and verification email. Custom attribute is created to identify the user's role as seen from the below figure; a user could be an “Admin” or a “Staff”.

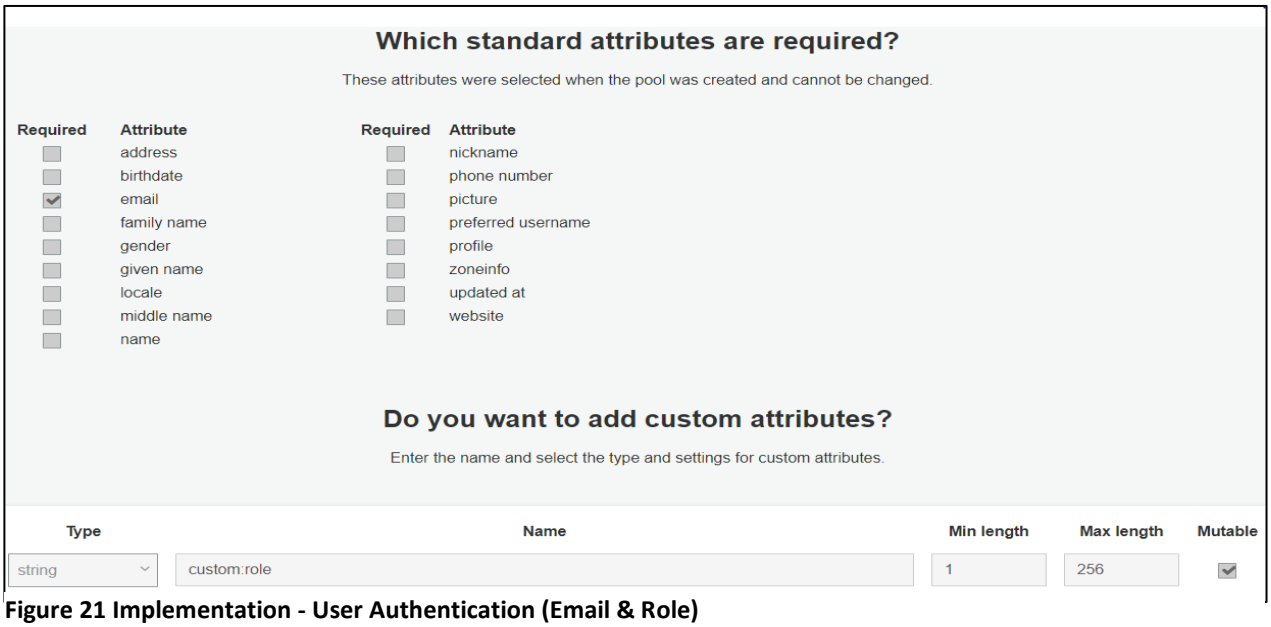


Figure 21 Implementation - User Authentication (Email & Role)

The new approach is to create an email from the web-app domain. This approach will demonstrate how professional the application is and portrait a high standard to the product. The below figure shows the used email for sending verification emails. It is worth to note, in order to use the email in Cognito service, the email needs to be registered in the domain first (in Amazon SES).

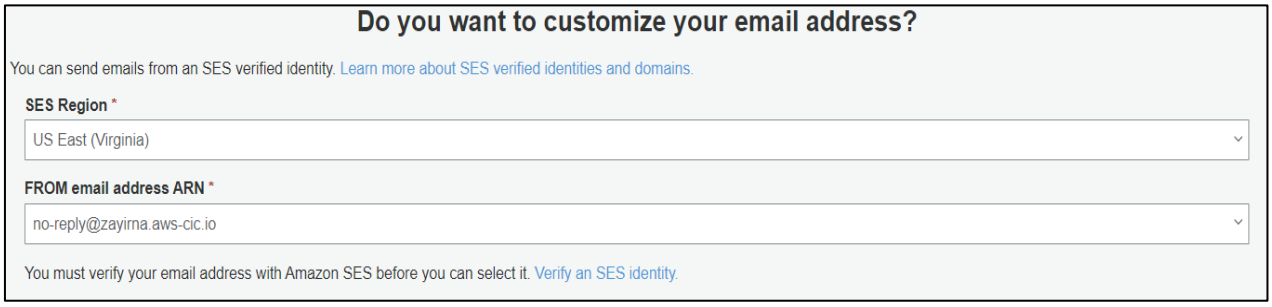


Figure 22 Implementation - User Authentication (Verfication Email - Final Approach)

The message's content has been altered to present a proper verification message and avoid being considered a suspicious email. A verification link was chosen instead of a code number to complete the verifying procedure. The below figure shows how the content of the email is written along with the email header.

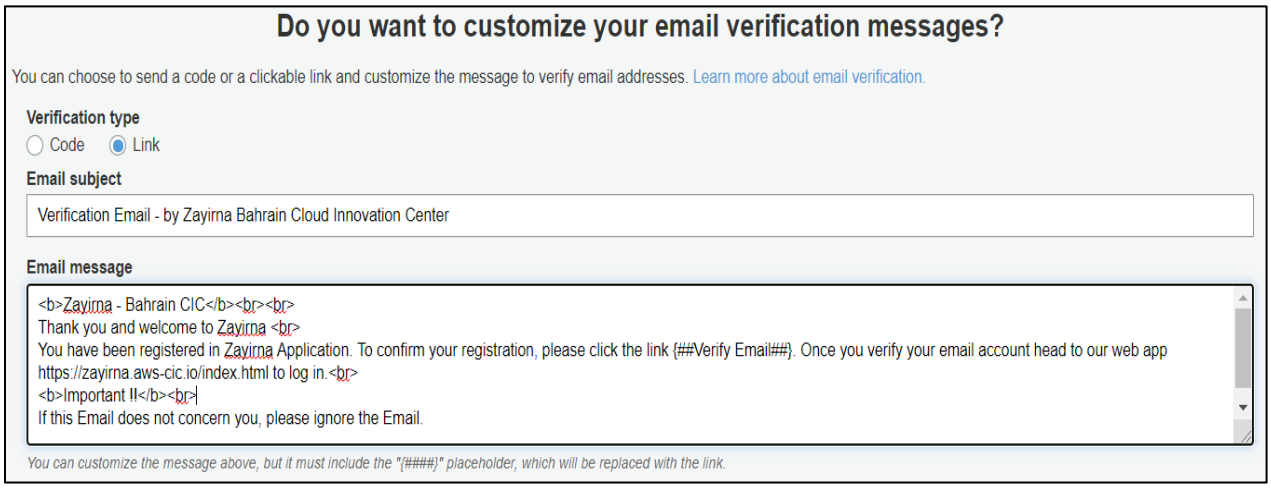


Figure 23 Implementation - User Authentication (Verfication Email - Message Content)

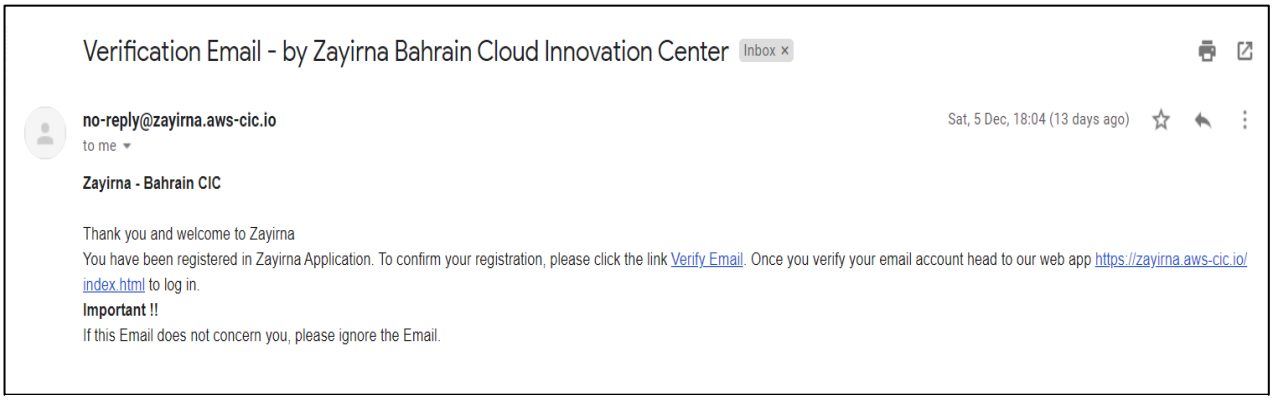


Figure 24 Implementation - User Authentication (Verfication Email - Email Content)

The first step to register an email using the domain name is to verify the SES service usage by the domain name. The below figure shows the form that needs to be filled in the SES email verification account. It is worth to note that the verification takes 48 hours to be completed.

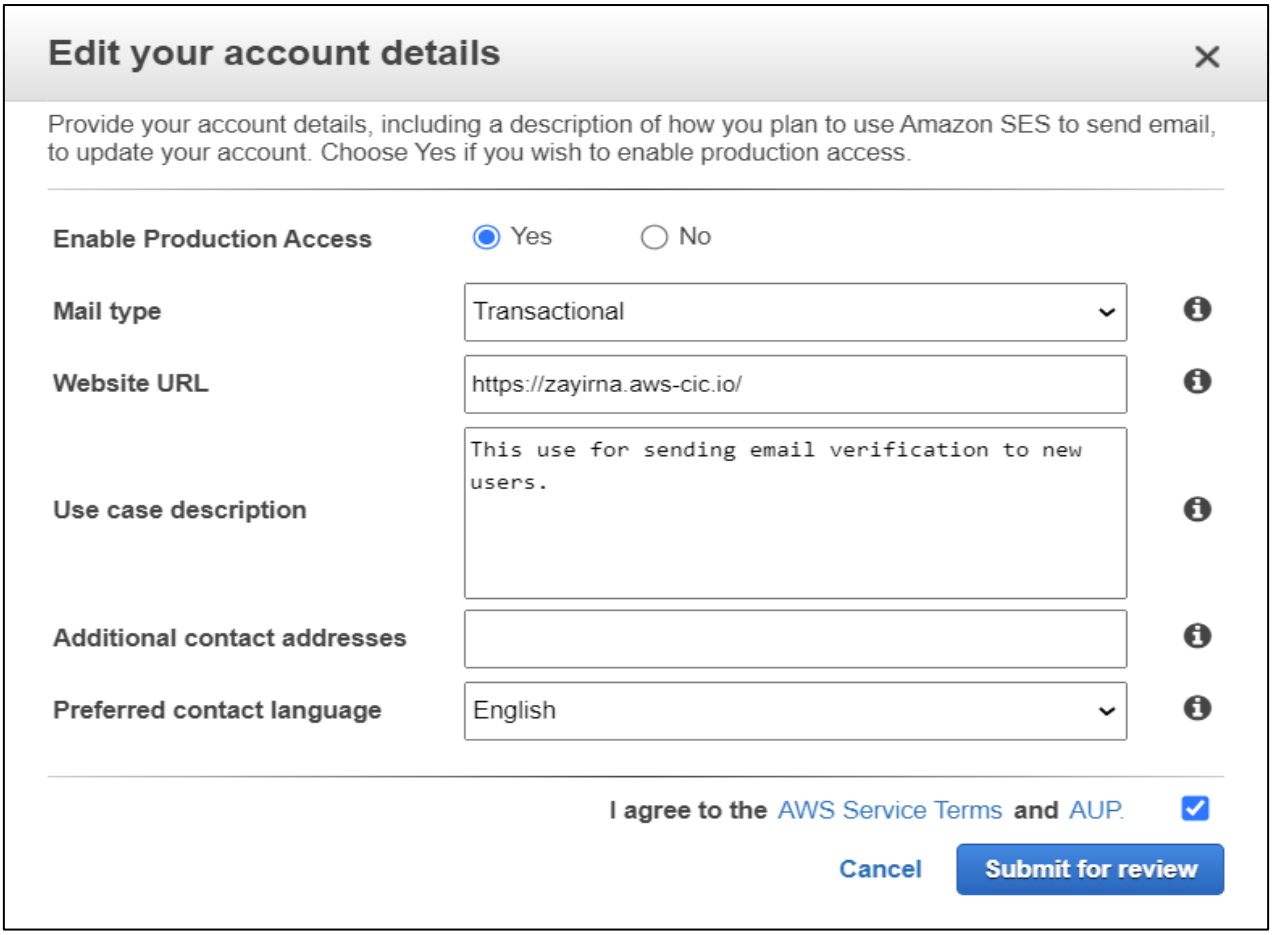


Figure 25 Implementation - User Authentication (Simple Email Service Verfication)

After the approval of SES usage, an email that uses the domain name needs to be registered. To perform this procedure, go to Amazon SES and create a new email address supplied with the product domain name as seen as the below figure.

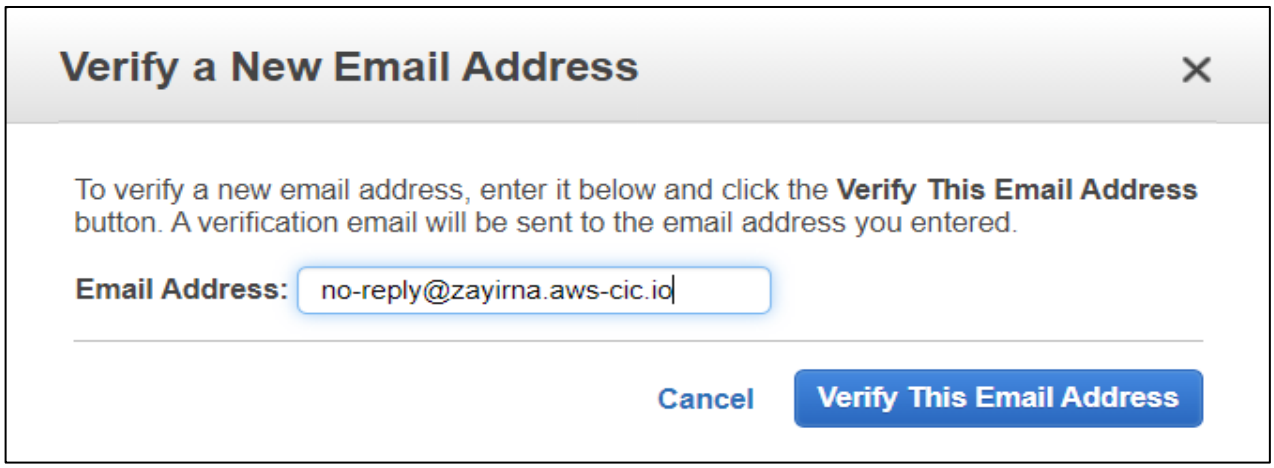


Figure 26 Implementation - User Authentication (Simple Email Service Verify New Email)

Once the above step is completed, it sends a verification link to that domain. Note, the receiving messages/emails for the domain was handled by the Amazon supervisor.

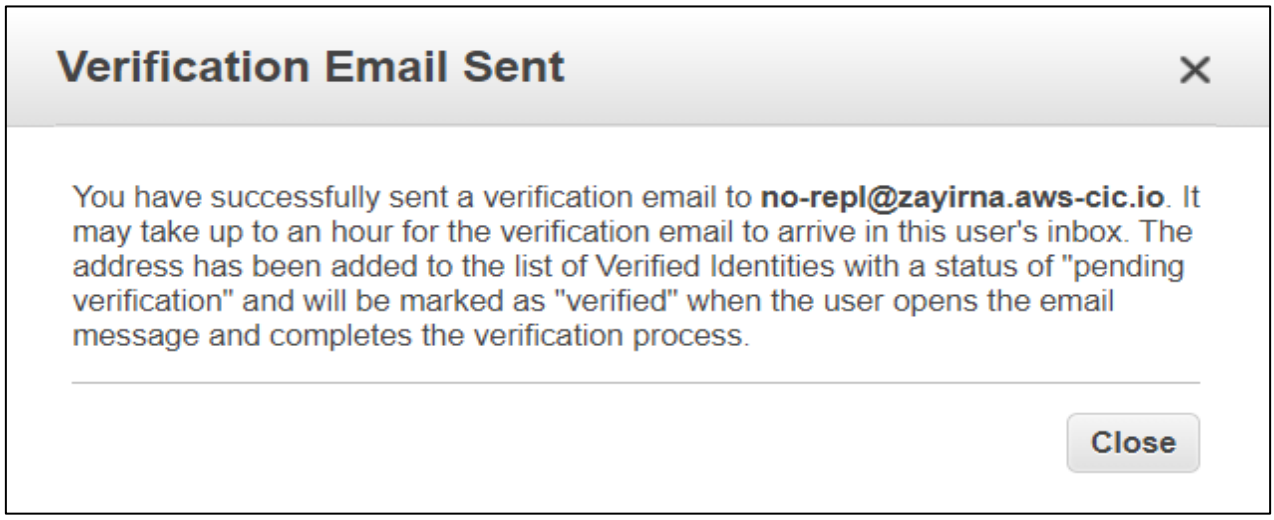


Figure 27 Implementation - User Authentication (Simple Email Service Verfiy Email Message)

Several functionalities in the web application use the cognition service (such as user login and registration). All the functionalities are discussed in detail in the appendix section under the Detailed Imeplentaion. Furthermore, the appendix section also includes a detailed explanation of the programming behind these functionalities.

- **Alternative approach**

Once the admin creates a new user, the Cognito sends an email to the new user. The previous approach was to use Amazon Cognito default email. This approach results with a suspicious email as seen from the below figure. This approach may give a bad reputation to the product as it does not portrait the demanded industry level.

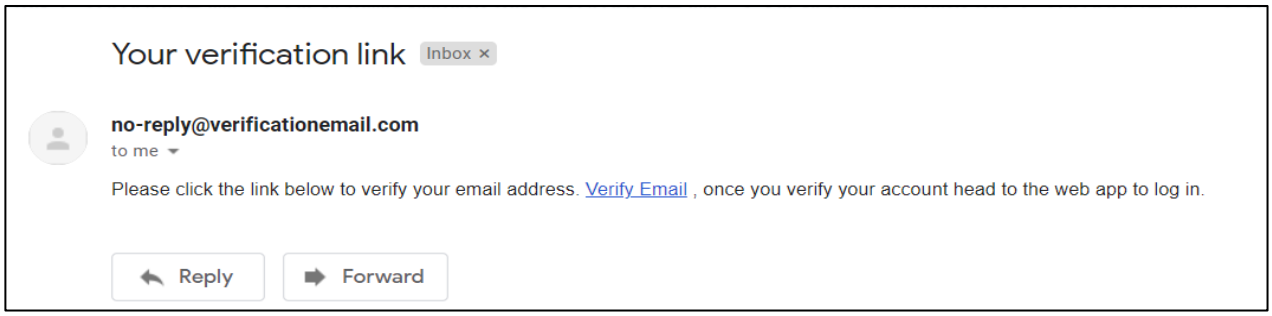


Figure 28 Implementation - User Authentication (Verfication Email - Previous Approach)

Another previous approach was to use the sign-in and sign-up page that Cognito service provides. This approach offers ready-made pages linked to the user pool in Cognito service; there is no need for SDKs. However, this approach was not followed as Cognito service does not provide the demanded customisation level needed to build a professional interface for the product.

3. ***Calculating Emotion Statistics***

This section includes the implementation of generating statistics of BTEA event visitors. The implementation contains creating a Lambda function to retrieve data (such as emotion and gender) and update statistic from a created DynamoDB table. This table will be populated until the end of the running event. Furthermore, another lambda function was designed to retrieve the final statistics from the DynamoDB table and store them in a JSON file. Following, an API was created to to read the JSON files (Emotion Statistics) and send the data to the browser for rendering.

- Step-By-Step Service Creation

The first step is to create a lambda that reads the face records from the s3 bucket and populate a DynamoDB table with the new statistics. This function is called “Store-Emotion-Statistics”.

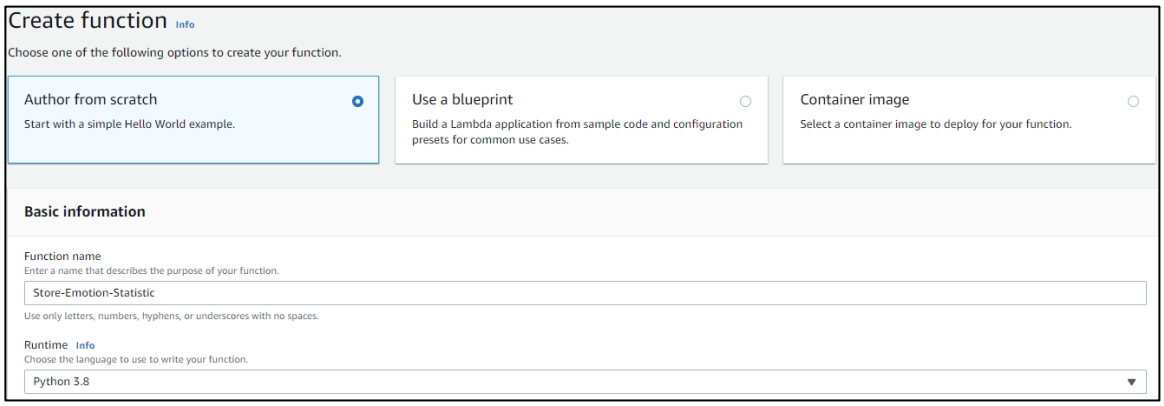


Figure 29 Implementation - Emotion Statistics (Store Emotion Statistics Lambda)

The second step is to create a DynamoDB table that will hold the statistics until the event is finished.

emotion	adultsEmotion	childrenEmotion	femaleEmotion	maleEmotion	totalEmotion	eventID
ANGRY	0	0	0	0	0	
CALM	0	0	0	0	0	
CONFUSED	0	0	0	0	0	
DISGUSTED	0	0	0	0	0	
FEAR	0	0	0	0	0	
HAPPY	0	0	0	0	0	
SAD	0	0	0	0	0	
SURPRISED	0	0	0	0	0	
eventID						<empty>

Figure 30 Implementation - Emotion Statistics (Emotion-Statistics DynamoDB Table)

The third step is to create another lambda that will retrieve the final statistics form the DynamoDB table and insert the data into a JSON file. This function is called “Store-Event-Emotion-Statistics”.

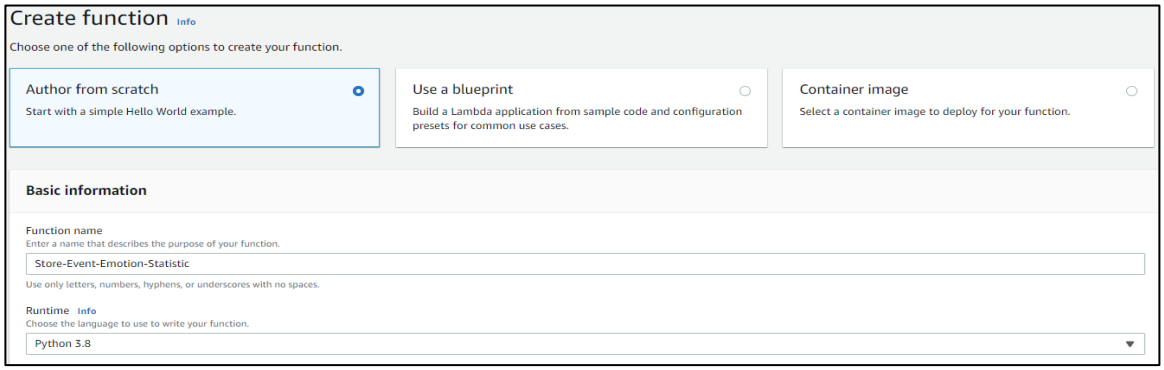


Figure 31 Implementation - Emotion Statistics (Store Event Emotion Statistics Lambda)

The fourth step is to create a RESTful API and lambda function that handles the API requests. The lambda function gets triggered once the API is called from the browser by the user. The following figures present the actions needed to create an API in the AWS console.

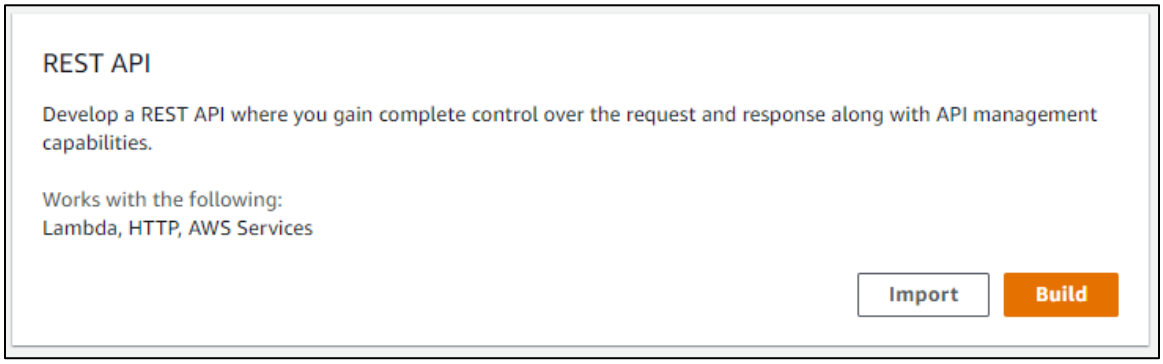


Figure 33 Implementation - Emotion Statistics (RESTful API)

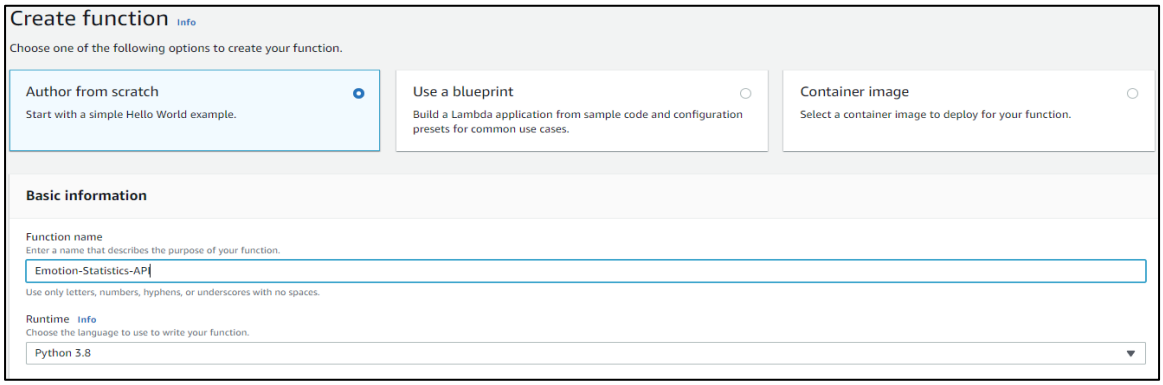


Figure 32 Implementation - Emotion Statistics (Emotion-Statistics-API Lambda)

The below figure shows the name and the endpoint type of the new API.

Figure 34 Implementation - Emotion Statistics (RESTful API Information)

After creating the API service, the next step is to create a method (GET method) and a resource name that will hold the method. The following figure shows where to create a resource.

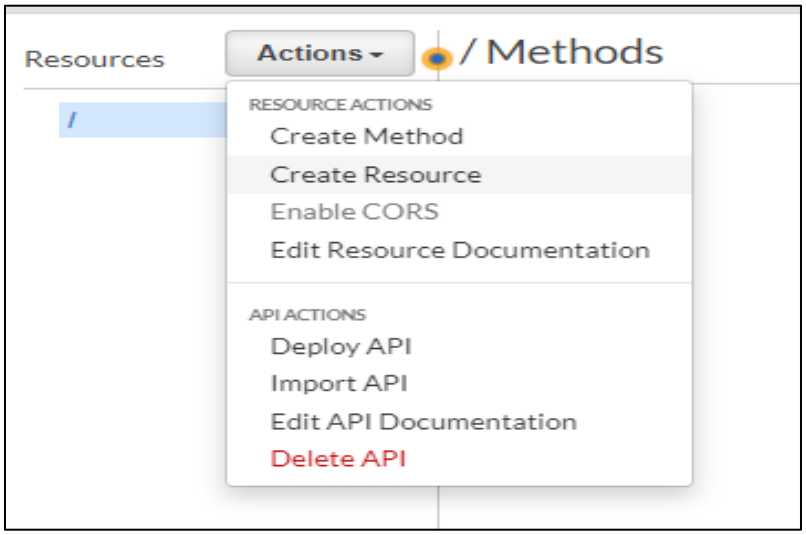


Figure 35 Implementation - Emotion Statistics (RESTful API - Build Resource 1)

The resource gets named by "/emotions" as the endpoint will contain the resource and the method name.

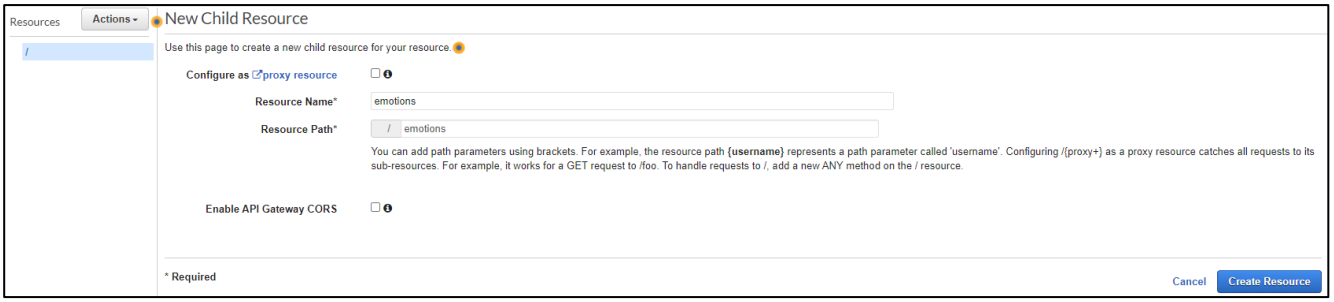


Figure 36 Implementation - Emotion Statistics (RESTful API - Build Resource 2)

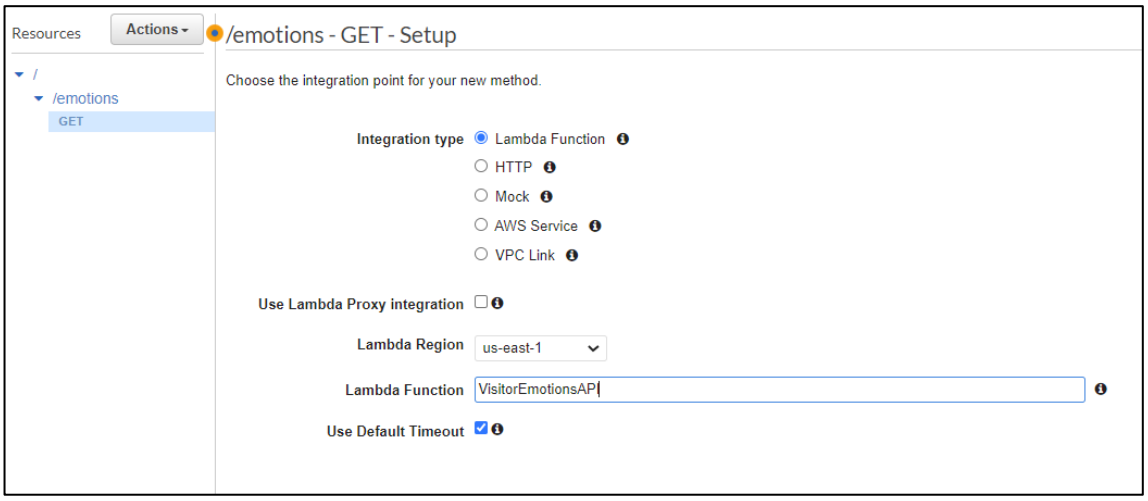


Figure 37 Implementation - Emotion Statistics (RESTful API - Buid GET method)

SDK needs to be generated to access the content of the API response. The following figure shows how to generate the SDK by going to "Stages" and under the "SDK Generation" choose JavaScript as it is the language used to read the SDK object. By generating and downloading the SDK in the same S3 Bucket as the web app, it is possible to read and render the emotion statistics to the user from the API.

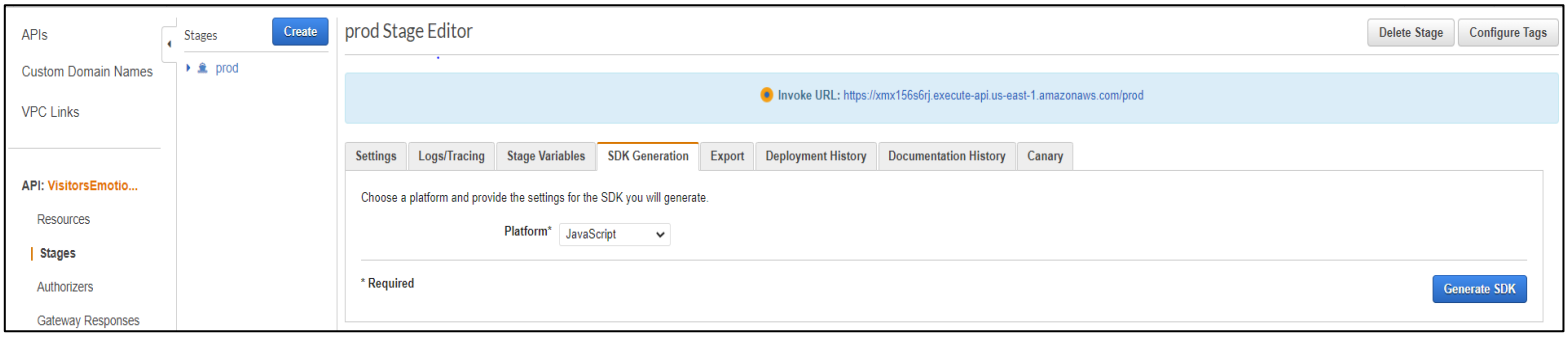


Figure 38 Implementation - Emotion Statistics (RESTful API – SDK Generation)

The Store-Emotion-Statistics and Store-Event-Emotion-Statistics lambda have several permissions that enable them to integrate with other services. Permissions are highlighted in detail in appendix section under the Detailed Imeplentaion. The appendix section also includes a detailed explanation of how Store-Emotion-Statistics and Store-Event-Emotion-Statistics lambda were programmed, and how emotion API was used.

- ## Alternative approach

There were several approaches for rendering the statistics in the browser. The previous approach was to open the JSON file from the browser as the statistics share the same storage unit (S3 Bucket). However, this approach is lengthy and complicated for a simple task due to hosting the application in an S3 Bucket with a static hosting behaviour. It is challenging to use scripting languages in a non-dynamic environment. Thus, the new approach overcame this obstacle using

AWS API Gateway (RESTful API) to retrieve/send the data to the browser and render the content using SDK provided by AWS.

Furthermore, generating statistics has also experienced different approaches. For instance, the previous method reads all the faces records once the event status changed to “Stopped”, which means the lambda is left with many faces to process. Therefore, this approach was not as efficient as it cost more. It takes a long time to read the massive face records in the data repository. The final approach operates in a real-time; it reads faces as soon as they get inserted in the data repository. This approach cost less as lambdas running duration scale was significantly reduced to seconds in comparing to minutes with the previous method.

4. ***Deeplens Heartbeat***

   This section involves keeping track of the Deeplens heartbeat. This process was accomplished by a Lambda function that retrieves the deeplens name and the occurrence time from the inserted face records of "Events" DynamoDB. After obtaining the demand attributes, they will be populated into another DynamoDB table that holds all the devices and the last processing time. Until the event stops, in real-time, a Lambda function created by one of the team members will continually read this DynamoDB table and render the content into the browser using Web-Socket API.

- Step-By-Step Service Creation

The first step is to create a lambda function called "Store-Deeplens-Status". This function gets triggered by the insertion of face records in Events DynamoDB table. It retrieves the name and the occurrence time of the captured face from records and updates the "Deeplens" DynamoDB with these two attributes.

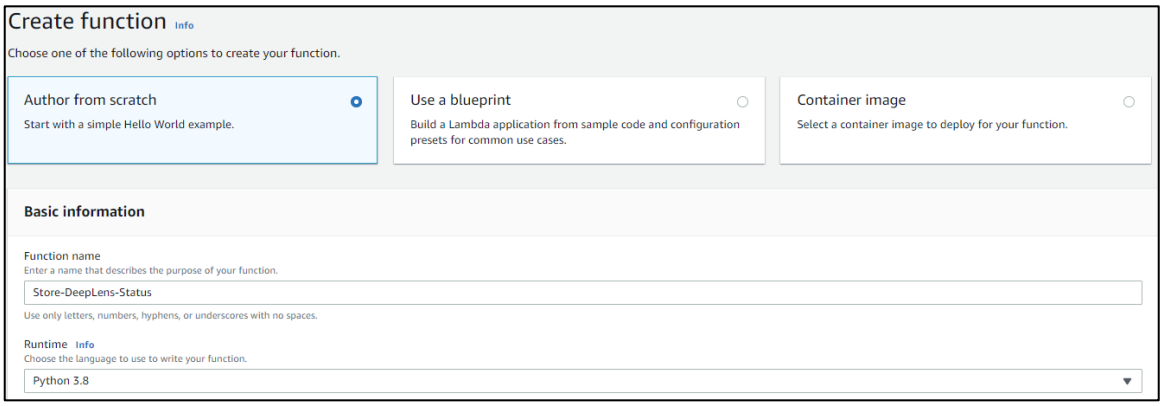


Figure 39 Implementation - Deeplens Heartbeat (Store Deeplens Status Function)

The next step is to create the "Deeplens" DynamoDB table that the above function will populate.

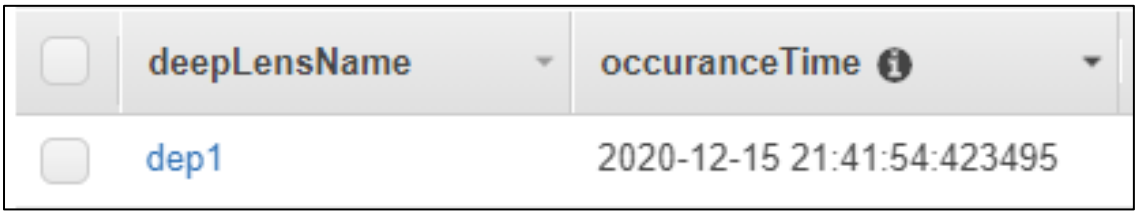


Figure 40 Implementation - Deeplens Heartbeat (Deeplens DynamoDB Table)

My colleague has created a Web Socket API. This API was used to transmit data in real-time to the cloud dashboard. Moreover, the Store-Deeplens-Status lambda has several permissions that enable it to integrate with other services. Permissions are highlighted in detail in appendix section under the Detailed Imeplentaion. Furthermore, the appendix section also includes a detailed explanation of how Store-Deeplens-Status lambda was programmed.

- ### Alternative approach

The previous approach was focusing on sending SNS messages from the Deeplens to a topic in the cloud. A programmed lambda listens to this topic (triggered) and fetches the data to "Deeplens" DynamoDB table. However, since Deeplens devices are running and fetching data non-stop, it is better to use these records' captured time instead of making cloud services listen to notifications that may jam under high traffic.

Testing

After implementing several approaches and methodologies, the solution needs to be tested against the system objectives, performance, and bugs. This section includes a description of all actions that took place to ensure the solution's effectiveness and efficiency. In addition, this section will contain the acceptance testing on the user's satisfaction with system functionalities.

I. Test Plan

Projects in the development environment are critical to be pushed directly to the production environment. Therefore, the testing environment is vital for project success and the reason behind conducting a test plan to determine the behaviour and the quality of the product. The following are the steps that were followed to test the product

- Test and ensure all system functionalities (functional and non-functional) are operating as planned.
- Perform a performance testing to ensure the performance scale is up to expectations as well as monitor the performance during a variety of scenarios.
- Perform a security test to protect critical data or operations from unauthorized access.
- Perform integration testing as in combining all the individual work as one final product.
- Test all created API for the web application.

II. Participants

Three participants were selected to experience and test the final solution; this approach helps gain different perspectives and insights that may not be experienced before. Also, this technique helps in exploring bug or issues that happened to be missed by the developers. Furthermore, participants were chosen based on their expertise in similar topics that were implemented in the project. The below table lists all participants' information (such as age, gender and background experience).

No.	Name	Age	Gender	Background
1	**Walaa Hasan**	30	Female	Walaa expertise in testing product migration between different environments. In addition, she has a rich experience in project integration as she ensures all parts done by individuals are been combined and build as one product. Therefore, from her standings, she is the most suitable participant for system functionality and integration testing.
2	**Salman Ahmed**	25	Male	Salman is an ICT graduated student that has a good experience in programming and building security measures for a web application. Also, he has a solid experience with APIs with their different methods. From his standings, he is the most suitable participant to test the security on the web application and the created APIs.

No.	Name	Age	Gender	Background
3	**Hassan Ali**	24	Male	Hassan is an ICT graduated student that has a few experiences in cloud security and performance checking. Also, he is an excellent web developer on different platforms. In addition, he has the skill to perform software testing. From his standings, he is the most suitable to test the performance of the solution as well as perform validation testing on forms and web pages.

Table 2 Testing Participants

III. Functionality Test Cases & Result

This part of the testing section involves a list of scenarios, these scenarios have been tested to ensure the achievement of the system functionality by examining the actual outcome and comparing it with the expected result. Thus, functionalities will then be determined by "Pass" or "Fail" status upon the actual outcome.

No.	Test Scenarios	Expected Result	Actual Result	Status
Face Recognizing & Segmenting				
1.1	Detect faces	Detect multiple faces from the Deeplens frames.	Detect multiple faces from the Deeplens frames.	Pass
1.2	Recognize faces & Determine Visitors' Redundancy	Recognize the detected faces. And identify a unique visitor if no similar faces were matching and identify a redundant visitor if a similar face were matching.	Recognize the detected faces. And identify a unique visitor if no similar faces were matching and identify a redundant visitor if a similar face were matching.	Pass
1.3	Identify Visitors' Gender	Identify visitors' gender by recognizing the faces.	Identify visitors' gender by recognizing the faces.	Pass
1.4	Identify Visitors' Age	Identify visitors' age by recognizing the faces.	Identify visitors' age by recognizing the faces.	Pass
1.5	Identify Visitors' Emotion	Identify visitors' emotion by recognizing the faces.	Identify visitors' emotion by recognizing the faces.	Pass
User Authentication & Registration				
2.1	User Login	User successfully log in and redirect to the dashboard of the web app.	User successfully log in and redirect to the dashboard of the web app.	Pass
2.2	Username & Password Validation	Validate username and password & display a proper message if empty or invalid.	Validate username and password & display a proper message if empty or invalid.	Pass
2.3	Register User	Admin shall register new users. Users shall receive a verification email.	Admin shall register new users. Users shall receive a verification email.	Pass
2.4	Validation for User Registration	Registration form is validated and displays proper messages in case of empty entries.	Registration form is validated and displays proper messages in case of empty entries.	Pass

No.	Test Scenarios	Expected Result	Actual Result	Status
2.5	Existing Username	Display invalid message ("Username already used") in case of used username.	Display invalid message ("Username already used") in case of used username.	Pass
2.6	Password Strength	Display a proper message for in case the password has a missing strength-criteria.	Display a proper message for in case the password has a missing strength-criteria.	Pass
2.7	Responsive Application	All pages in the web app adjust upon the user screen.	All pages in the web app adjust upon the user screen.	Pass
Performance & Security				
3.1	Face Detection & Recognition	Faces are detected are recognized from captured the frame in less than three-to-four seconds.	Faces are detected are recognized from captured the frame in less than three-to-four seconds.	Pass
3.2	Page Rendering	Pages render to the user in less than five seconds.	Pages render to the user in less than five seconds.	Pass
3.3	Accessing Pages	Only logged in users can access the dashboard. If user is not logged in, they are redirected to the login page.	Only logged in users can access the dashboard. If user is not logged in, they are redirected to the login page.	Pass
3.4	Authorizing Users to Access Create Account Page	Only users with "ADMIN" role are authorized to enter the registration page. If the user is not an "ADMIN", they are redirected to the dashboard.	Only users with "ADMIN" role are authorized to enter the registration page. If the user is not an "ADMIN", they are redirected to the dashboard.	Pass
Integration & API Testing				
4.1	Deeplens Device and Trigger Integration	By alteration of scheduled events' status, the deeplens gets triggered and start to detect faces.	By alteration of scheduled events, the deeplens gets triggered and start to detect faces.	Pass
4.2	Deeplens Device and Real-time Dashboard Integration	The count and the segmentation of detected faces are integrated with a real-time dashboard to present statistics.	The count and the segmentation of detected faces are integrated with a real-time dashboard to present statistics.	Pass
4.3	Emotion Statistics and Event Integration	By alteration of scheduled events' status, the emotion statistics gets finalized and form an overall statistic for the respective event.	By alteration of scheduled events' status, the emotion statistics gets finalized and form an overall statistic for the respective event.	Pass
4.4	Emotion Statistic API	Emotion statistics API retrieves all the concluded events and sends their emotion statistics to the browser without errors.	Emotion statistics API retrieves all the concluded events and sends their emotion statistics to the browser without errors	Pass

Table 3 Functionality Test Cases & Result

IV. Acceptance Tests Process & Results

This section of the testing plan involves participants acceptance of the overall product. This testing provides perspectives of the participants and determines if a certain procedure has failed or passed. The following table contains the process and the result of acceptance testing.

No.	Participant	Process	Result
1	Walaa Hasan	Schedule new event from the dashboard.	Deeplens starts to capture frames and detect faces upon the start of the scheduled event.
			The deeplens system integrates with real-time subsystem and render live counting and segmenting statistics.
2	Salman Ahmed	Attempt to access the dashboard without being logged in.	Gets redirected to the login page.
		Attempt to access user registration page being not admin.	Gets redirected to dashboard page.
3	Hassan Ali	Submit empty form and wrong credentials in the login page.	“Missing Credentials” message displays for empty form and “Invalid Credentials” message for invalid entries. The total number of attempts were three times or wait 15 minutes if exceeded number of attempts.
		Submit empty form and existing username the registration page.	“Missing Credentials” message displays for empty form and “User already exists” message for already registered username.

Table 4 Acceptance Tests Process & Results

V. Usability testing results & statistics

The objective of the performed usability testing was examining participants’ feedback on the performance of the product, the ease of use, and issues or bugs. The testing implicates face detection and recognition, user authentication, account registration, and lastly is emotion statistics.

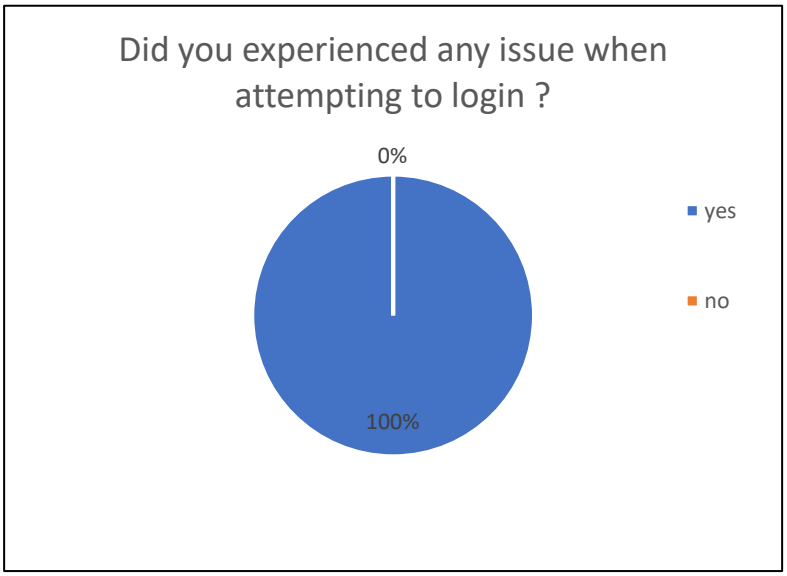


Figure 41 Usability Testing - Login Statistics

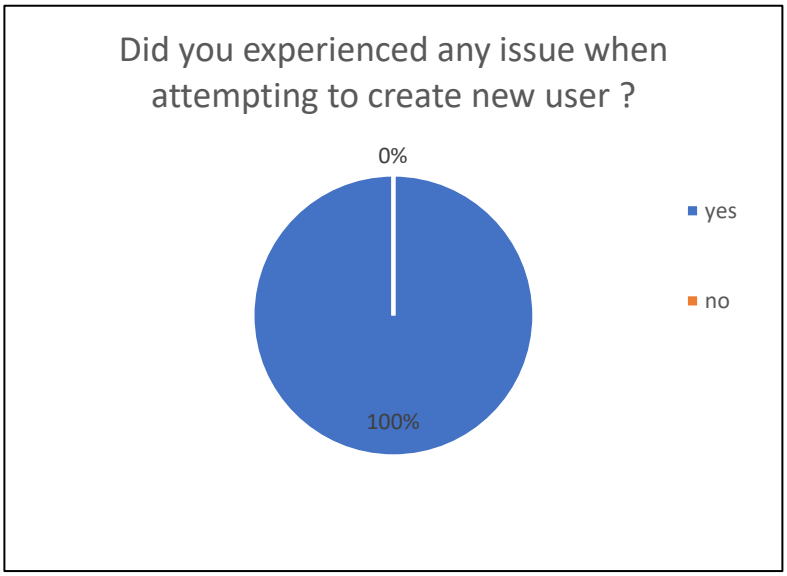


Figure 42 Usability Testing - Registration Statistics

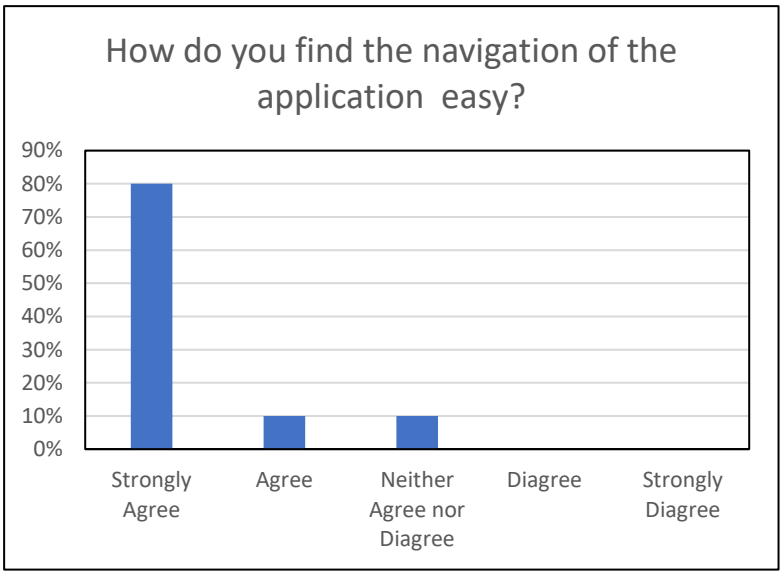


Figure 43 Usability Testing - Web App Navigation Statistics

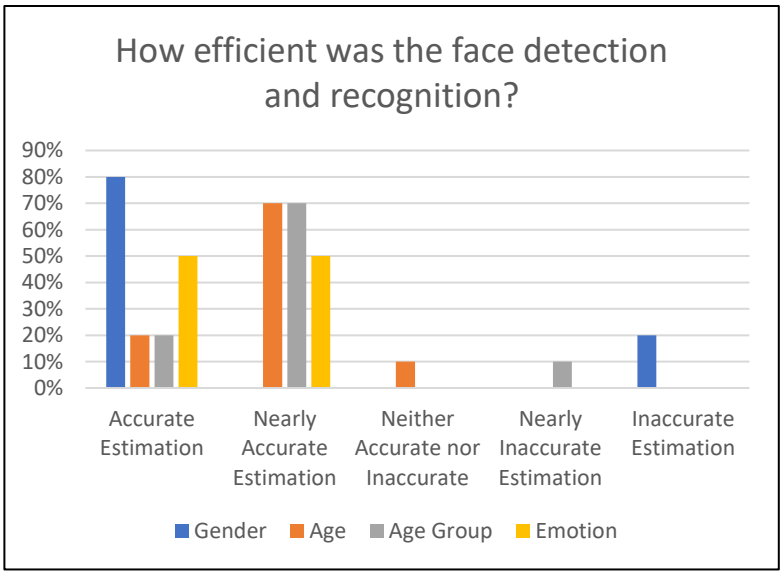


Figure 44 Usability Testing - Face Detection & Recognition Statistics

According to the survey results, as seen from the above charts, participants experienced no issues in logging in and signing up process (figure 74 & 75). Thus, the survey reults identifies that the authentication and registration feature were successfully implemented. Furthermore, the survey results on the web app navigation (figure 76) were mostly easy with 90% in total who agrees. This indicates that the navigation was not that complex to the users. Statistics on the face detection & recognition process recorded 80% on accurate gender estimation whereas 20% of the participants have experienced an inaccurate estimation. Moreover, the participant had a good experience in age and age group estimation with overly 90%. Whereas, the emotion statistics, indicates a great result with 50% accurate and nearly accurate estimations.

Discussions, LESPI & Conclusion

This section of the paper summarized the functionalities and achieved the final solution's objectives and reflects on the experience and issues that arose during the project development. This section will also highlight possible aspects that could be improved, and that contributed to the Kingdom of Bahrain.

I. System Functionality

The final solution has achieved all the critical objectives of this project. This project aims to count and segment visitors of BTEA events accurately while displaying real-time statistics (done by colleagues) on a web app dashboard. The solution also contains a final pdf file that holds final statistics for the individual event and the functionality to schedule, update and delete events (done by colleagues). Furthermore, this automated solution achieved these functionalities (refer to the Requirements section under the Methodology) by using an intelligent camera and various cloud services provided by Bahrain AWS. The camera detects and recognizes faces from video footage and creates one record per recognized face. These records are handled by cloud services that extract data and generate statistics on visitors (such as gender, age and emotion). These statistics are sent in real-time to a cloud dashboard for client visioning. The dashboard has several functionalities, including scheduling new events and view past event statistics (done by colleagues); all dashboard functionality requires an authorization procedure. The project has achieved a high secure authentication and authorization using cloud service provided by Bahrain AWS. Moreover, as seen from the methodology section of this paper, these functionalities' implementation has followed the exact design. The deployment and infrastructure design showcase how the solution's component is integrated and how the system functionality was achieved.

II. Summary of Achieved Objectives

The final solution has accomplished several objectives that solved client issues. In contrast, the final solution eliminates the manual system by deploying an automated AI system that simultaneously provides an accurate counting and segmenting of BTEA's visitors. In addition, this product accomplishes other aspects that the previous system lacks. It reduces the cost and increases the productivity of staff by focusing on other tasks. Furthermore, a team of four members was responsible for project completion. Therefore, this paper will only include sections that were in my responsibility range. View my colleagues' thesis papers in case needing an abroad perspective. Moreover, the below table lists all objectives and a description of how they were achieved.

No.	Objectives	Status	Description
1	An automated solution that eliminates the manual system	Achieved	Deployed Amazon Deeplens camera on the entrances and exits of an event. The camera captures video footage for analysis process and eliminates the manual procedure (staff uses clickers to count visitors).
2	Reduce Cost	Achieved	The final product cost of the utilized services was cheap; pay-as-you-go. This approach minimizes the operation cost

			as cloud services are cheap compared to the previous manual system's used equipment and human resources
3	Accurate Visitor Segmenting	Achieved	Each detected face by Deeplens cameras goes under the recognition process. This process index faces and extract the facial features of the respective face. Data such as gender and age were identified using Rekognition service that AWS provides. In addition, the segmentation process includes the identification of visitors' redundancy; checks if the visitor is unique or redundant.
4	Security (Highly Secured Product)	Achieved	Several approaches have contributed to achieving more secured product • Deeplens cameras require a password to grant access to the device; Individuals cannot access the deeplens via SSH without obtaining the password. • Extracted data from visitors' faces are stored in an encrypted repository in the cloud; minimizing the chances of misusing data by attackers if they were obtained. • All used services have been granted the minimal privileges to perform their respective task; read or write privileges are only granted to services that need to perform these actions. • Dashboard user authentication & authorization (another objective).
5	Dashboard Security	Achieved	The login and sign-up procedure in the cloud dashboard was achieved by AWS service. The procedure enables users to sign in securely as the credentials are secure and protected by Amazon Cognito service. Furthermore, all pages require authentication and user authorization before rendering the pages.
6	Generating Statistics (Emotions)	Achieved	From facial features of the detected faces, emotions data are extracted to generate statistics for the event. The statistics were categorized based on three main inputs; they are gender, age group and the emotion itself. The statistics are displayed on the dashboard once the event has ended.
7	User Verification	Achieved	Once the admin creates a new user account, a verification email is sent to the respective user. Email cloud service handles the verification procedure and is responsible for sending a verification message.
8	Deeplens Monitoring	Achieved	Troubleshooting Deeplens camera is essential. The project achieved the objective to monitor the Deeplens heartbeat by rendering the last time the device has operated on the cloud dashboard.

Table 5 Achieved Objectives

To conclude, project implementation has achieved all the objectives that were documented in the project plan. The implementation was extended by additional features that were not in scope as these features enhance the overall product and fulfil client's satisfaction. Thus, the constraints (such as time and cost limitations) of the project did not affect the project's progress.

III. Project Issues

The project implementation was the most critical task in building the product. This phase had numerous issues and challenges that lead to change the approaches and designs. Furthermore, due to specific services, the implementation had some restriction in approaches. This section will discuss the issues and limitation as well as propose the solutions that resolved the problem.

No.	Issues / Limitations	Resolved Method
1	Face detection & recognition is not in real-time	The first approach of detecting faces was to send the whole captured frame to the cloud service for detection and recognition. This approach took 30-40 seconds to detect and recognize a single frame. Due to this issue, a trained model provided by AWS was used. The model is trained to detect faces and return the probability and dimensions of faces. It was possible to crop the image and resize the frame to a specific a single face. This approach minimized the size of the file that was needed to be sent to reckognition service and shorten the time needed to detect and recognise a single face.
2	Server-side Scripting	Hosting the web application in an S3 Bucket resulted in limiting the scripting languages. JQuery, PHP, Python, AJAX and many more scripting languages were not possible to use. However, Java Script was the only option to resolve the issue, since it runs on the client browser. However, JS had its limitation when it comes to handling requests. For instance, files or APIs were not possible to read with just JS. As a solution, SDKs were used to read files and API and communicate with cloud services similar case in User Authentication & Authorization part in the implementation section.
3	Internet Connection	The internet latency was the most concern issue with the face detection and recognition part. The product was tested with a variety of Wi-Fi connections. As a result, a proper connection is needed to maintain steady performance. However, the programming approach has been altered to recognize the top four detected faces in a single frame. This approach enables fast detection as no more than four faces are queued to be recognized and extract their facial features.

4	Query Tables	The storage environment of the product was mainly in NoSQL database (DynamoDB). The partition key needs to be known to query a table whereas querying tables on specific attributes (not a partition key) was not possible. A scan operation is performed on the whole table to match the condition with the known value to resolve the issue. As an improvement, indexes can be used to provide good retrieval performance. However, the scanned tables do not hold a vast number of records.
5	Fault Classification in Rekognition Service	This issue is that Amazon Rekognition service doest not return 100% correct recognition values. For instance, it recognizes a female face as a male. This issue arose since the trained model that AWS uses does not apply to Arab traditional clothing. The proposed solution is to build and train a model that is capable of recognizing female and male individuals that wears “Thob”, “Abaya” or “Niqab”.

IV. Future Work

This section will include suggestions that would be beneficial to the next version of this product. Changes that would enhance and stabilized the product to be reliable in the industry and the production environment are

1. *Web Application Hosting*
   The web application of the final solution is hosted in an S3 Bucket. This approach limits the solution as it S3 Bucket doest not support scripting languages. In case the application gets hosted in a proper environment, many functionalities could be added, and other existing functionalities could be rebuilt in a professional implementation standard.

2. *Web Application Framework*
   Using a framework would ease the web application building. For instance, Laravel framework provides robust features, elegant syntax and a simple and secure authentication process. Another example is to use WordPress to build and design web application.

3. *Deeplens Recognition Model*
   Since the final solution faces issues in recognition, a trained model would help recognize faces. In addition, if a trained model is used, the amount of time needed to detect and recognize a single face would be minimized compared to the current solution.

4. *Upgrade Emotion Statistics*
   The current emotion statistics lack the depth considering the available data. For instance, emotion statistics could be generated from the age group based on gender with the available data. In contrast, the current solution separates the

statistics of gender and the age group. It would also be more analytic to consider the peak of specific emotions and showcase them to the client.

5. *Social Distancing & Mask Detection*
   This proposed idea will only be applicable during the COVID crisis, detecting if a visitor is wearing a mask is essential for the event's organizers and the visitor himself. Keeping safety measures is vital; therefore, calculating social distance is vital to maintain the visitors' safety and staff.

6. *Generating Peak Time Areas/Zones Reports of Visitors*
   Event organizers will benefit from reports that determine where visitors are mostly attending and which categories of the booth mostly attract visitors. These statistics are compelling and have a significant impact on business analytics when they are known. Addressing the peak time is excellent data for event organizers as they will prepare and have precautions of a high number of visitors.

V. Synopsis of my experience

The experience with Bahrain AWS had great potential. The experience was rich in learnings from many perspectives. A vast amount of knowledge was gained as various AWS services were utilized. However, the used services were a fraction of what they provide. Therefore, I believe this experience will be beneficial to my future career as it helped build a solid structure in the cloud computing field. It would be a requirement for employees to have knowledge in this cloud computing field. I am grateful to be one of the chosen to participate in this experience. This experience uncovered my love for emerging technologies and unleashed my potential to brainstorm new solutions and ideas. In addition, this experience allowed me to grow my skill and characteristic as an individual. Since it was a group-based project, my communication skills have grown more robust due to regular meetings. Exchanging ideas and perspectives was a delightful experience as not only a person could gain more knowledge but also learn from his mistakes.

Furthermore, it was a memorable experience as it was the first time I have experienced and felt that I am implementing a real project that would be pushed to production in case of success. Nevertheless, this experience came with harsh and stressful situations. Many ideas were implemented and were altered as it was not an excellent approach. Various of prerequisite knowledge were demanded before beginning the implementation. Even while implementing the knowledge never stopped incrementing, these requirements took many hours to study and grasp a basic understanding. Therefore, this experience has built a decent knowledge structure for emerging technologies seekers.

Moreover, besides the AWS experience, I learnt how to manage an entire project by myself. Gratitude to the Project Management Course, I was able to study, plan, design, implement, test and monitor a whole project. In addition, I was able to determine the project's requirement while considering how to satisfy the client and match their expectations.

VI. Bahraini Perspectives

Bahrain aims to attract international events that could strengthen its position as tourism attraction on the regional and international level. Targeted events that Bahrain aims to engage with a vast number of tourists. Therefore, maintaining a considerable number of attendees is a complicated procedure, especially as the population and tourist number increments over time. Furthermore, this situation's major problem is obtaining an accurate statistic on the number of attendees and their segmentation. Statistics are an essential requirement for Bahrain in reflecting tourism attraction with other countries. Bahrain could solve this critical issue by deploying an emerging technology that utilized cloud services. The solutions use an AI technology that counts the number of attendees and differentiate between them. This approach results in an accurate number of visitors and their segmentation. The solution includes a cloud dashboard that presents real-time (live) statistics of the running exhibitions. In addition, when an event concludes, a PDF document gets generated containing the exhibition statistics while event organizers can visualize the emotion statistics with the archived records. Statistics are beneficial to Bahrain as the accurate tourist/attendees' number reflects the progress the kingdom is experiencing. These statistics provide opportunities that increase the economy and flourish businesses in the public and private sector. The solution will make 3,000 – 4,000 BHD profit to exhibition organizers by saving money on equipment, staff and overall used approach in the manual procedure (refer to plan document). It is worth to note that the profit varies depending on the usage of the solution.

Moreover, the proposed solution has a potential demand by various sectors as the solution has numerous application fields. For instance, mall/shop owners could make use of this product. Generating statistics of malls is a way to understand the consumer (buyers). From the resulting statistics, owners may change or improve their respective aspects to increase their profit rate. From Bahrain perspectives, this product fits perfectly in "2030 Vision". It improves Bahrain's diversification of the economy and provides opportunities for business owners to grow and strengthen the market. Simultaneously make Bahrain focuses on developing the tourism industry with emerging technologies instead of the manual used approaches.

VII. Conclusion

In conclusion, extensive knowledge was gained from building the final product. Aspects such as cloud services and on-edge computing were the major acquired knowledge areas. The experience and the gained insights on AWS cloud services mainly resulted from researching particular services and understanding their actual usage. Trial and error were the followed method to experience cloud services. The more research is done, the more AWS amazes me with their capability on emerging technologies. AWS has a mass number of services that they provide for the public. The "Rekognition" services were the most astonishing in comparison to other used services. This AI service can detect and recognise faces, objects, and text in images or video streams.

Moreover, exploring and researching other similar services that other cloud providers provide will extend the gained knowledge. Professional certificates in cloud computing will widen the scale of expertise. The final product has a variety of application areas. Concerts, Malls, airports are all scenarios where the product will result in rich statistics. In essence, any public place where civilians gather will be an excellent application area. The product definitely can be improved in several aspects as it is the first version prototype. Hosting the web application in a dynamic environment or building and training a face recognition model are all room for improvements. With skilled and expert individuals, the product could be ready for production by the next version.

Furthermore, this product's practical implications are solving a business problem in counting and segmenting visitors of an event. This product resolves this problem by replacing the old manual system with an automated AI cloud system. This solution eliminates the need for staff and equipment as well as results with accurate data.

References

Graaf, S. V. (2018, February 13). Doorman Project - Hackathon. Retrieved November 17, 2020, from https://aws.amazon.com/deeplens/community-projects/Doorman/

Bek, Sebastian & Monari, Eduardo. (2016). The crowd congestion level — A new measure for risk assessment in video-based crowd monitoring. 1212-1217. 10.1109/GlobalSIP.2016.7906034.

Hara, Tadayuki & Severt, Kimberly & Shapoval, Valeriya. (2016). Estimating Total Number of Attendees to an Open Free Non-Gated Outdoor Cultural Event – A Case of Zora! Festival in Eatonville, Florida, USA. Journal of Tourism Economics, Policy and Hospitality Management. 3. 1-16.

DAVIES, L., RAMCHANDANI, G. and COLEMAN, R. (2010). Measuring attendance: issues and implications for estimating the impact of free-to-view sports events. International Journal of Sports Marketing and Sponsorship, 12 (1), 11-23.

Jagirdar, Srinivas & Venkata, K & Reddy, Subba & Qyser, Dr. (2013). CLOUD COMPUTING BASICS. International Journal of Advanced Research in Computer and Communication Engineering. 1. 343.

Shallal, Qahtan & Bokhari, Mohammad. (2016). CLOUD COMPUTING SERVICE MODELS: A COMPARATIVE STUDY. IEEE Network. 16-18.

Fernando, A. (2017, November 16). Is AWS Lambda yet another Platform as a Service? Retrieved November 20, 2020, from https://medium.com/99xtechnology/is-aws-lambda-just-another-platform-as-a-service-7c7a1998e786

Tariq, Itrat & Mufti, Tabish. (2020). Face Detection And Recognition. https://www.researchgate.net/publication/344737834_FACE_DETECTION_AND_RECOGNITION

Hazim, Nawaf & Al-Dabbagh, Sinan Sameer Mahmood & Esam Matti, Wael. (2016). Face Recognition: A Literature Review. International Journal of Applied Information Systems. 11. 21-31. 10.5120/ijais2016451597.

Petters, J. (2020, June 17). AWS, Azure, Google: Cloud Services Comparison - Varonis. Retrieved November 21, 2020, from https://www.varonis.com/blog/aws-vs-azure-vs-google/

AWS. (2018, May). What is AWS in globe. Retrieved November 21, 2020, from https://aws.amazon.com/what-is-aws

Butler, B. (2016, July 13). Who's got the best cloud latency? Retrieved November 21, 2020, from https://www.networkworld.com/article/3095022/who-s-got-the-best-cloud-latency.html

Fernandes, J. (2017). Introduction to AWS DeepLens. Retrieved November 21, 2020, from https://aws.amazon.com/deeplens/

Hendrix, R. (2014). Lambda. Retrieved November 22, 2020, from https://aws.amazon.com/lambda/

Akiwatkar, R. (2017, August). Simform. Retrieved November 21, 2020, from https://www.simform.com/aws-lambda-vs-ec2/

Mishra, A. (2019). Machine learning in the AWS Cloud: Add intelligence to applications with Amazon SageMaker and Amazon Rekognition. Retrieved November 21, 2020, from https://aws.amazon.com/rekognition/?blog-cards.sort-by=item.additionalFields.createdDate

Bobriakov, I. (2018, October 01). Comparison of Top 6 Cloud APIs for Computer Vision. Retrieved November 21, 2020, from https://medium.com/activewizards-machine-learning-company/comparison-of-top-6-cloud-apis-for-computer-vision-ebf2d299be73

Rangel, D. (2015). DynamoDB: Everything you need to know about Amazon Web Service's NoSQL database. Retrieved November 22, 2020, from https://aws.amazon.com/dynamodb/

Jayendrapatil. (2017, March 20). Jayendra's Blog. Retrieved November 22, 2020, from https://jayendrapatil.com/aws-storage-options-rds-dynamodb/

Rouse, M. (2018, July 31). AWS S3 bucket - Definition from WhatIs.com. Retrieved November 22, 2020, from https://searchaws.techtarget.com/definition/AWS-bucket

Sosnowski, M. (2020, May 06). The pros and cons of using AWS Cognito for user authentication • intent. Retrieved November 22, 2020, from https://withintent.com/blog/the-pros-and-cons-of-using-aws-cognito-for-user-authentication-in-web-and-mobile-apps/

Roose, H. (2015). Cognito. Retrieved December 07, 2020, from https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-user-pool-settings-ses-authorization-to-send-email.html

Atbasoglu, C. (2014). SES. Retrieved December 05, 2020, from https://aws.amazon.com/ses/

Janakiram, M. (2015, August 06). Five Reasons to Consider Amazon API Gateway for Your Next Microservices Project. Retrieved December 07, 2020, from https://thenewstack.io/five-reasons-to-consider-amazon-api-gateway-for-your-next-microservices-project/

Solutions, M. (2017, October 03). Python: 7 Important Reasons Why You Should Use Python. Retrieved December 07, 2020, from https://medium.com/@mindfiresolutions.usa/python-7-important-reasons-why-you-should-use-python-5801a98a0d0b

Paul, J. (2020, February 07). What is Python Good for? – 10 Reasons to Learn Python in 2020. Retrieved December 07, 2020, from https://hackernoon.com/10-reasons-to-learn-python-in-2018-f473dc35e2ee

Hosting, F. (2019, September 26). The Advantages of Using JavaScript. Retrieved December 07, 2020, from https://www.futurehosting.com/blog/the-advantages-of-using-javascript/

Subhajit. (2018, July 05). Advantages of Using JavaScript in Web Technologies. Retrieved December 07, 2020, from https://dev.to/javablog/advantages-of-using-javascript-in-web-technologies-1c4g

Walsh, K. (2011, October 26). Kelly Walsh. Retrieved December 07, 2020, from https://www.emergingedtech.com/2011/10/creately-is-an-easy-to-use-easy-access-affordable-web-based-diagramming-tool/

CloudCraft, P. (2015). Pricing. Retrieved December 07, 2020, from https://www.cloudcraft.co/pricing

Group, O. (2017, April 12). The best prototyping tools. Retrieved December 07, 2020, from https://medium.com/@order_group/the-best-prototyping-tools-8d7dc5c8ee27

Sweeney, A. (2019, September 03). The Must Read HTML vs CSS Infographic. Retrieved December 07, 2020, from https://www.codingdojo.com/blog/html-vs-css-inforgraphic

Jayaram, M. (2016, June 10). Crowd Counting Through Head Detection. Retrieved December 08, 2020, from https://flex.flinders.edu.au/file/b4a679b1-1faf-4002-a1a2-1a47e4ae401a/1/Thesis%20report%20Crowd%20Counting.pdf

Roqueiro, Damian & Petrushin, Valery. (2007). Counting people using video cameras. IJPEDS. 22. 193-209. 10.1080/17445760601139096.

Siregar S, Syahputra M & Rahmat R (2018, February 01). Human face recognition using eigenface in cloud computing environment. Retrieved December 09, 2020, from https://iopscience.iop.org/article/10.1088/1757-899X/308/1/012013

Halligan, N. (2019, August 15). Life on cloud Amazon. Retrieved October 09, 2020, from https://www.arabianbusiness.com/technology/425818-on-cloud-amazon/

Gilroy, M. (2020, May 14). Comparing Cloud Platforms for Machine Learning Applications. Retrieved October 09, 2020, from https://mc.ai/comparing-cloud-platforms-for-machine-learning-applications/

Banerjee, P. (2020, June 26). Top 5 Programming Languages and their Libraries for Machine Learning in 2020. Retrieved October 09, 2020, from https://www.geeksforgeeks.org/top-5-programming-languages-and-their-libraries-for-machine-learning-in-2020/

Hare, C. (2019, October 11). A Five-Minute Overview of AWS Rekognition. Retrieved October 09, 2020, from https://medium.com/@labrlearning/a-five-minute-overview-of-aws-rekognition-562b34a885fc/

Ersoy, P. (2020, August 06). New Generation Computer Vision: AWS DeepLens. Retrieved October 09, 2020, from https://towardsdatascience.com/new-generation-computer-vision-aws-deeplens-45052e39b4bc/

Just Landed, S. (2014, September 10). Internet in Bahrain. Retrieved October 09, 2020, from https://www.justlanded.com/english/Bahrain/Bahrain-Guide/Telephone-Internet/Internet-in-Bahrain/

Patra, C. (2019, September 06). Amazon DynamoDB: What it is and what you really should know. Retrieved October 09, 2020, from https://cloudacademy.com/blog/amazon-dynamodb-ten-things/

Arias, D. (2018, May 03). Adding Salt to Hashing: A Better Way to Store Passwords. Retrieved October 09, 2020, from https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/

Smallcombe, M. (2019, June 11). Amazon Quicksight: Overview and Review. Retrieved October 09, 2020, from https://www.xplenty.com/blog/amazon-quicksight-overview-and-review/

Mishra, S. (2019, August 14). The importance of prototyping in designing. Retrieved October 24, 2020, from https://uxdesign.cc/importance-of-prototyping-in-designing-7287c7035a0d

Fowler, A. (2017, June 13). 10 Advantages of NoSQL over RDBMS. Retrieved December 12, 2020, from https://www.dummies.com/programming/big-data/10-advantages-of-nosql-over-rdbms/

Guthrie, G. (2020, October 22). Everything you need to know about architectural diagrams (and how to draw one). Retrieved December 12, 2020, from https://cacoo.com/blog/everything-you-need-to-know-about-architectural-diagrams-and-how-to-draw-one/

Hindman, B. (2017, September 16). How to Make Your Events Memorable - Run of Show Weekly. Retrieved November 28, 2020, from https://splashthat.com/blog/emotional-marketing-for-events

Visual, P. (2016). What is Deployment Diagram? Retrieved November 07, 2020, from https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-isdeployment-diagram/

Lynch, W. (2019, June 26). UML from What to How with Use Case and Activity Diagram. Retrieved December 13, 2020, from https://medium.com/@warren2lynch/uml-from-what-to-how-with-use-case-and-activity-diagram-61758068b52a

Larman, C. (2005, March 04). Informit. Retrieved December 13, 2020, from https://www.informit.com/articles/article.aspx?p=360441

Green, A. (2019, March 11). The Advantages of an Interview Over a Questionnaire. Retrieved December 14, 2020, from https://bizfluent.com/info-8220458-advantages-interview-over-questionnaire.html

Appendices

I. *Appendix I: System and User Manuals*

This section presents the manual for login and registration procedure as well as how to display the emotion statistics of a particular event in the cloud dashboard.
Web Application URL: https://zayirna.aws-cic.io/
Username: Nicloas@123 ***Password:*** Nicloas@123

a) User Manual - Login

Once the welcome page renders (Using the above URL), a login button will be visible to redirect to the login page as seen from the below figure.

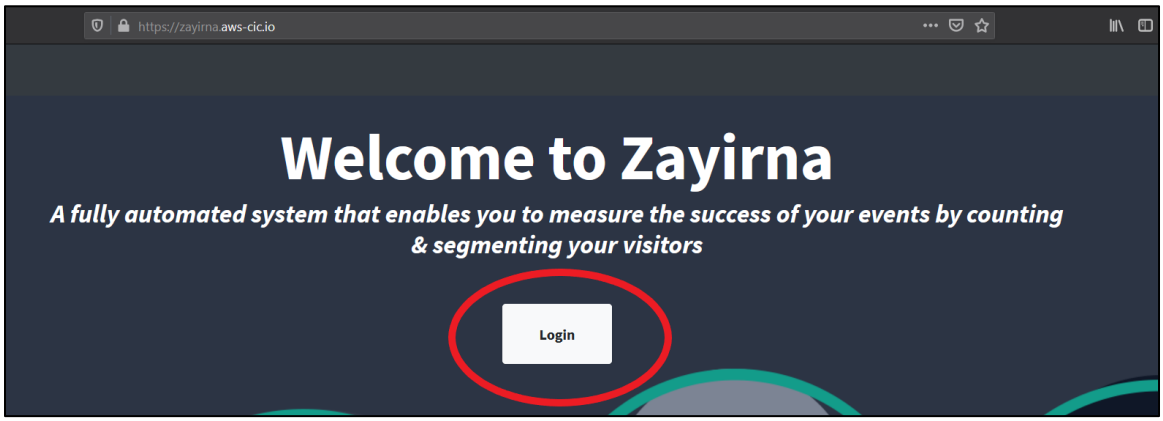


Figure 41 System & User Manual - Welcome page

Once the user clickes the "Login" button, he/she will be redirected to the login page. The login page will be similar as the below figure.

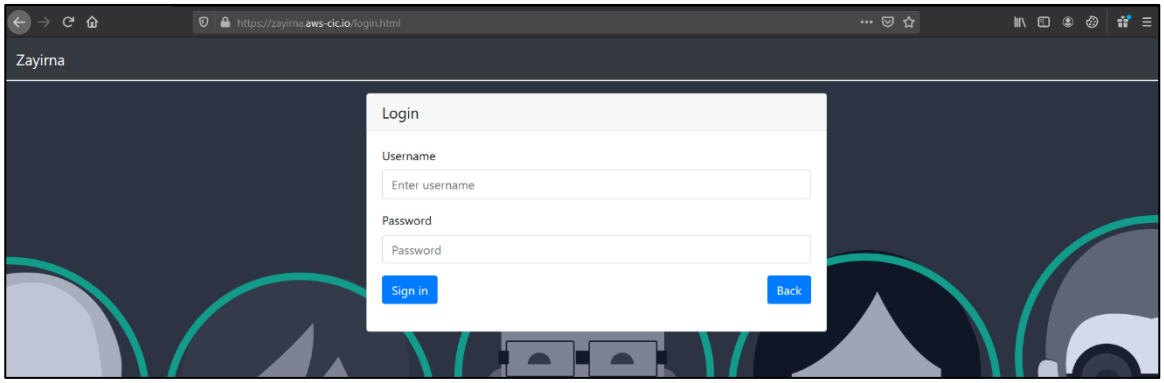


Figure 42 System & User Manual - Login page

The authentication system was built with a solid validation handling, error messages will be visible once the entries were invalid. It is worth to mention the user is limited with three attempts, incase the user pass the limited attempts, they must wait for 15 minutes

to try again. The below figure displays the validation message if the user clicked the “sign in” button without entring the credentials.

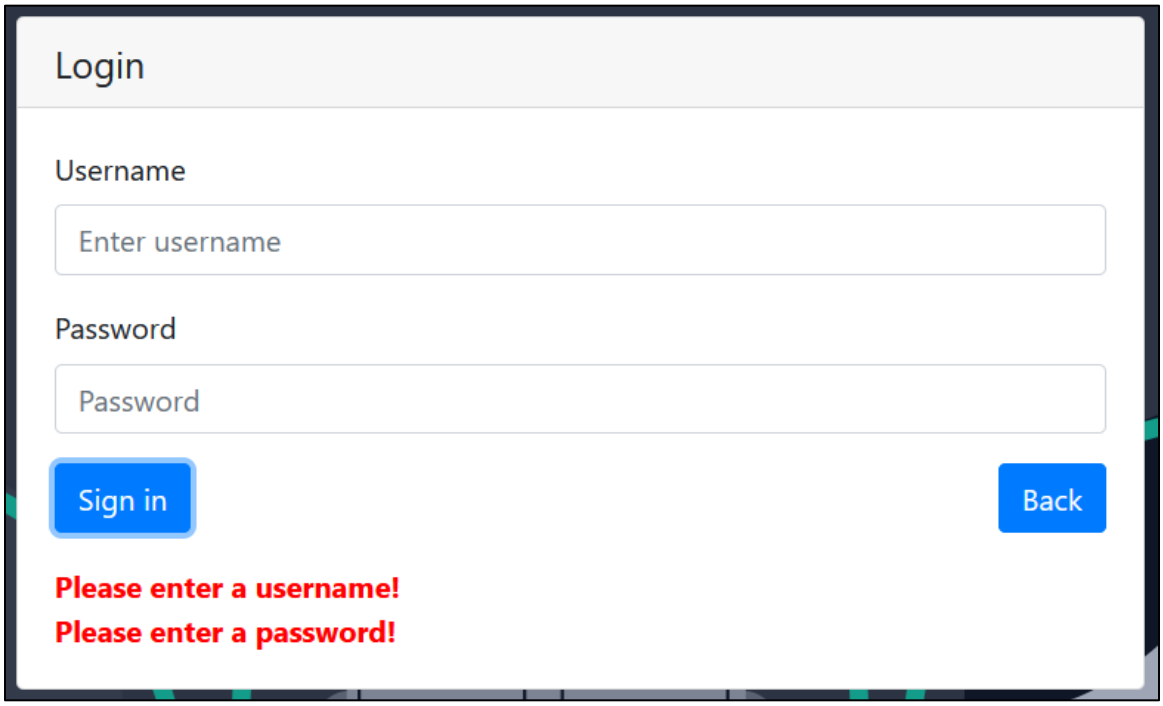


Figure 43 System & User Manual - Login Missing Entries

The below figure displays the validation message if the user clicked the “sign in” button with invalid username or password.

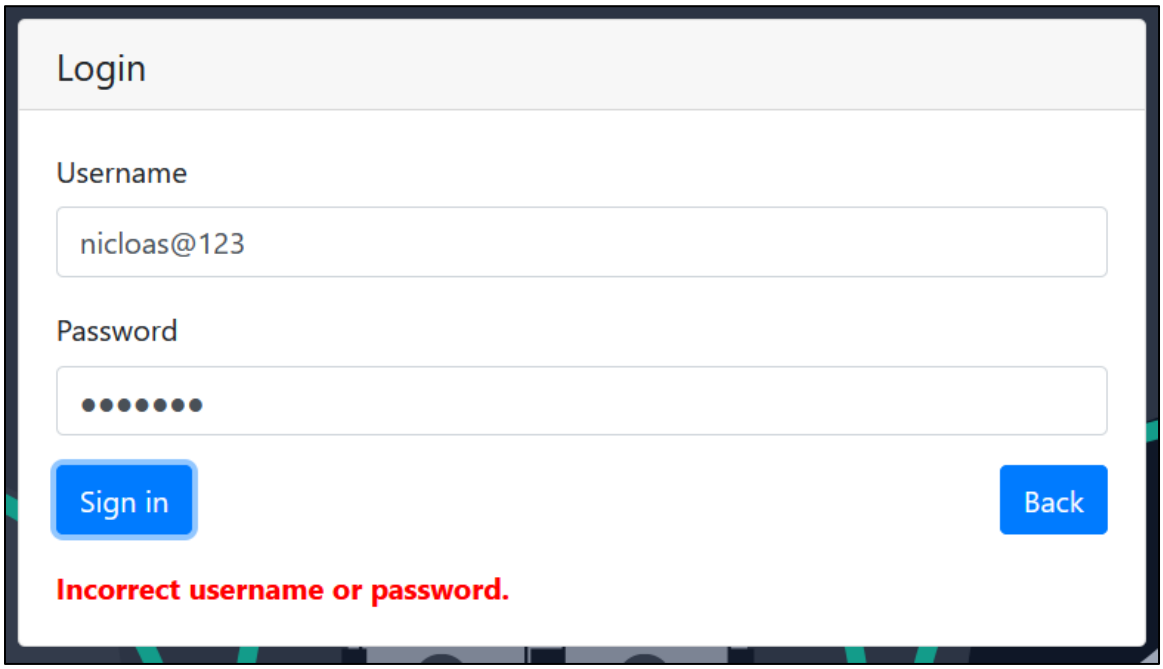


Figure 44 System & User Manual - Login Invalid Password

The authentication system will notify the user about his/her last attempt as a warning, this procedure occurs in the third attempt.

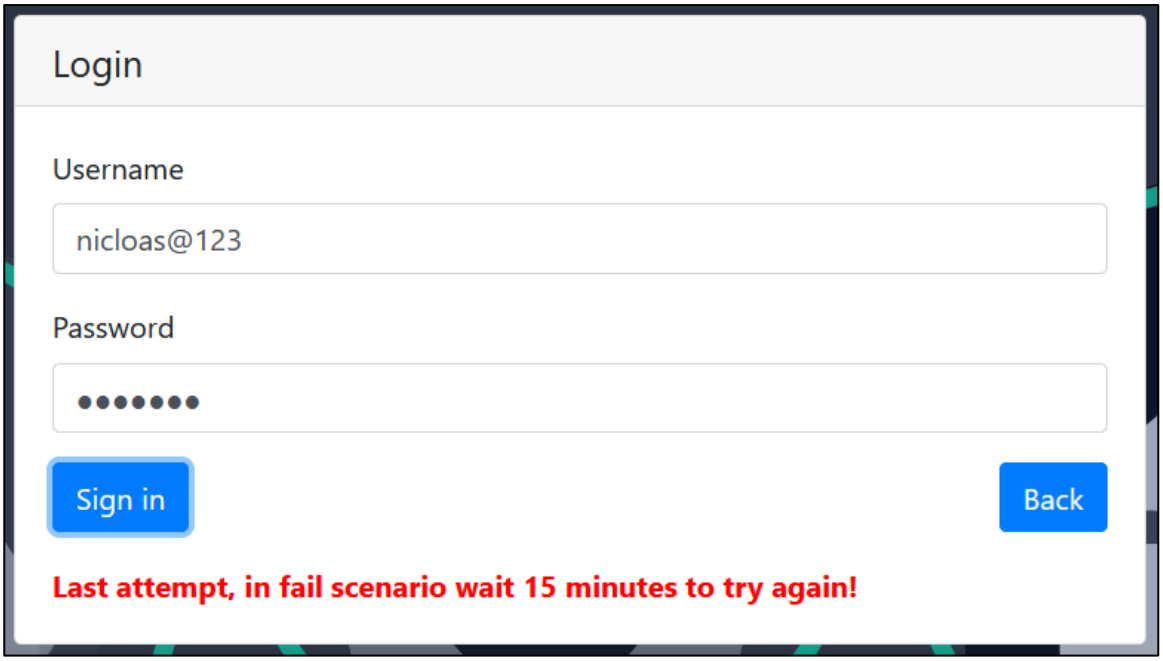


Figure 45 System & User Manual - Login Third Attempt

If the third attempt recorded an invalid log, a waiting message will be displayed for the user. The next attempt gets postpond by 15 minutes.

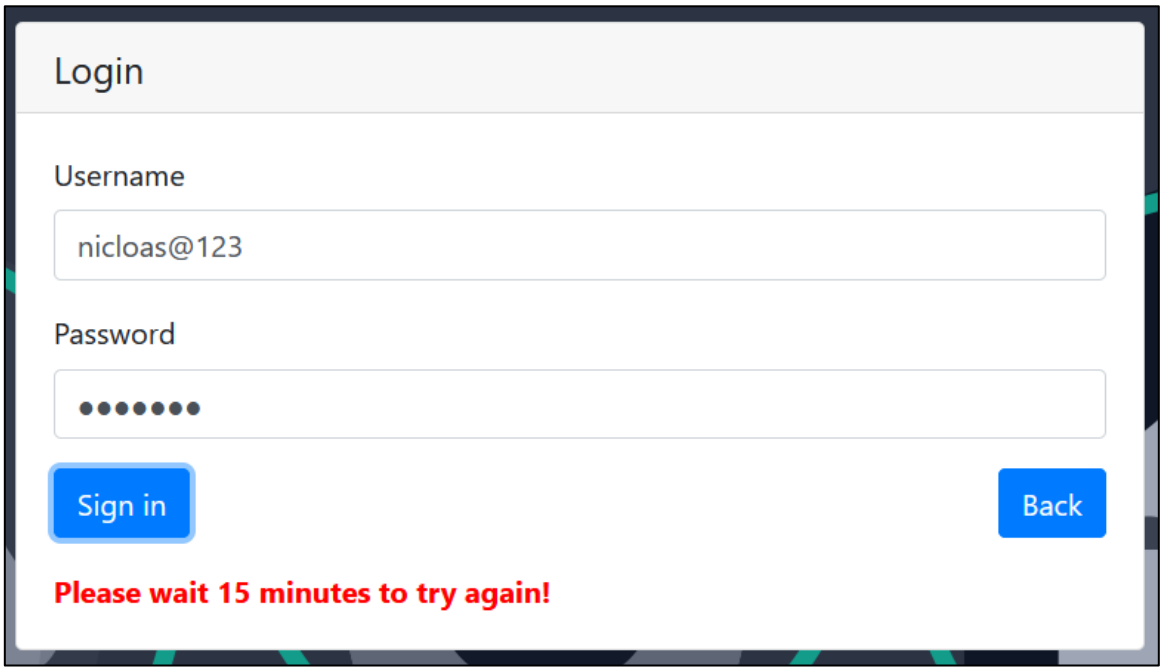


Figure 46 System & User Manual - Login Invalid Third Attempt

In case the credentials were valid, the user gets redirected to the dashboard page as seen from the below figure, the username is displayed at thr top right side of the page.

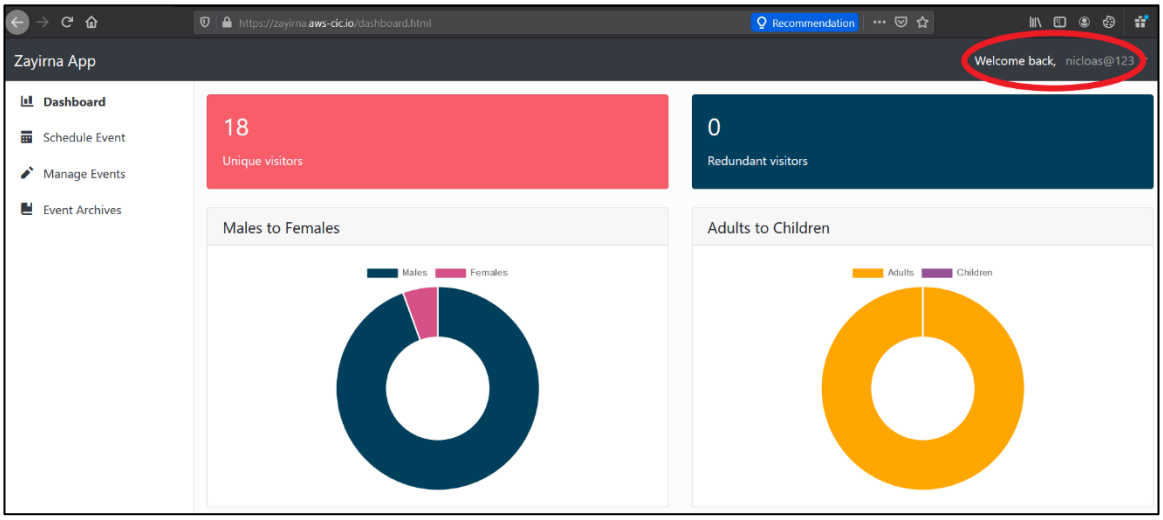


Figure 47 System & User Manual - Dashboard Page

b) User Manual - Emotion Statistics

To display emotion statistics for a particular event, the user must navigate to “Event Archive” that is located at the left side tab as seen in the below figure.

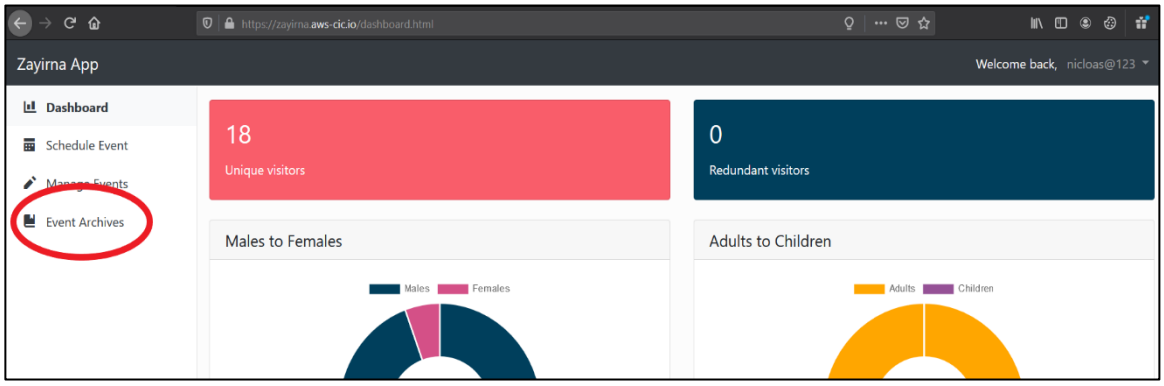


Figure 48 System & User Manual - Archived Events

The archived page displays all the concluded events along with their statistics. Noted that this page is work of one of my colleagues. The below figure is a screenshot of where the emotion statistics page can be accessed, this section is located at the bottom of the "Arhieved Events" page.

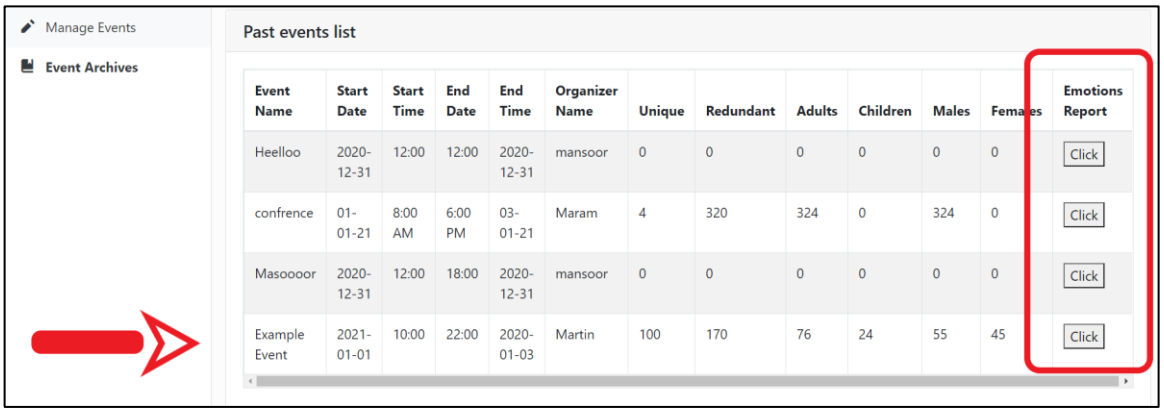


Figure 49 System & User Manual - Archived Events List

From the above figure, it can be seen that for every record has a button to redirect the emotion statistic page of that respective event. For demonstration purposes, "ExampleEvent" event is selected.

Once the user clicks on the button they will be redirected to the "Visitor-Emotion" page. This page contains statistics of visitors' emotions based on gender, age group and the emotion itself as seen from the below two figures.

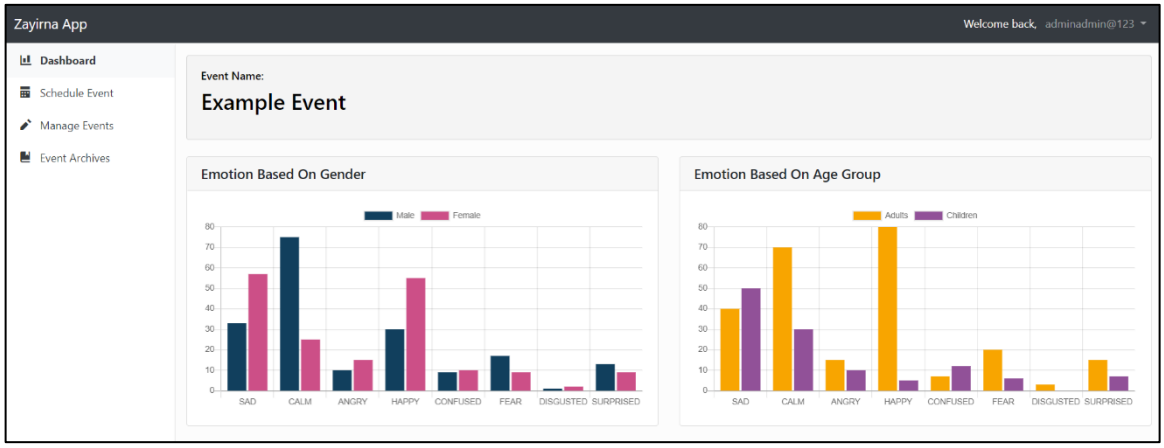


Figure 50 System & User Manual - Visitor Emotion Page (Gender & Age Group Statistics)

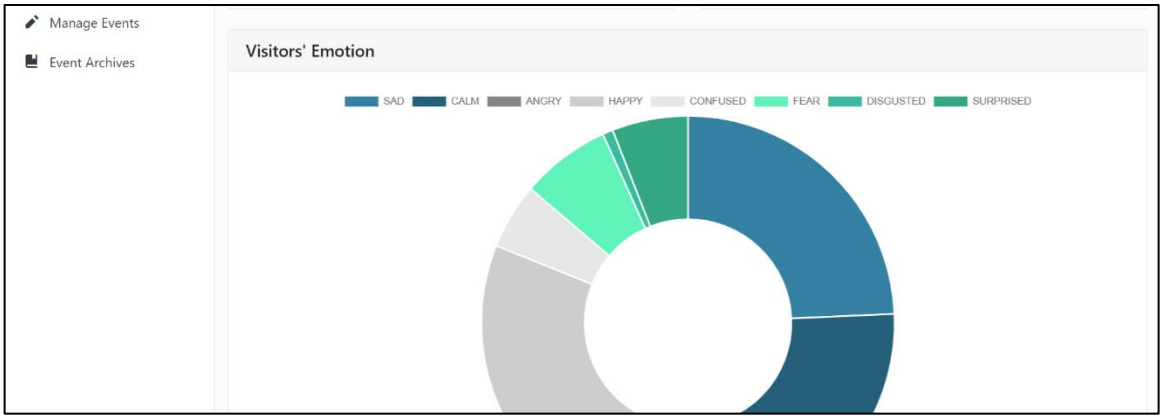


Figure 51 System & User Manual - Visitor Emotion Page (Overall Emotion Statistics)

c) Admin Manual - Registration

The registration procedure can only be performed by a user with an "Admin" role, if the respective user has a "Staff" role, they are not able to access the registration page or as well as the dashboard will not display the tabs that are only restricted to "Admin" users.

The tab that is responsible to redirect the admin to the registration page can be found under the username. The below figure displays the location surrounded by red circle.

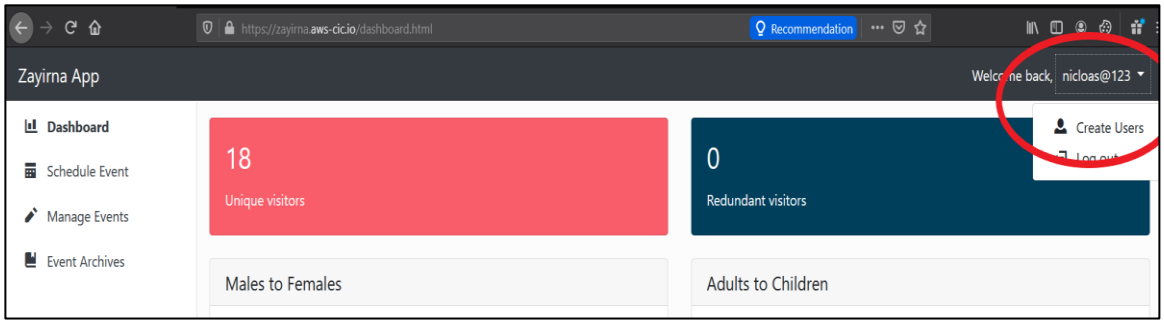


Figure 52 System & User Manual - Create User Tab

Once the user clicks on "Create Users" tab, he/she will be directed to the registration page. The below figure presents how the registration page looks like.

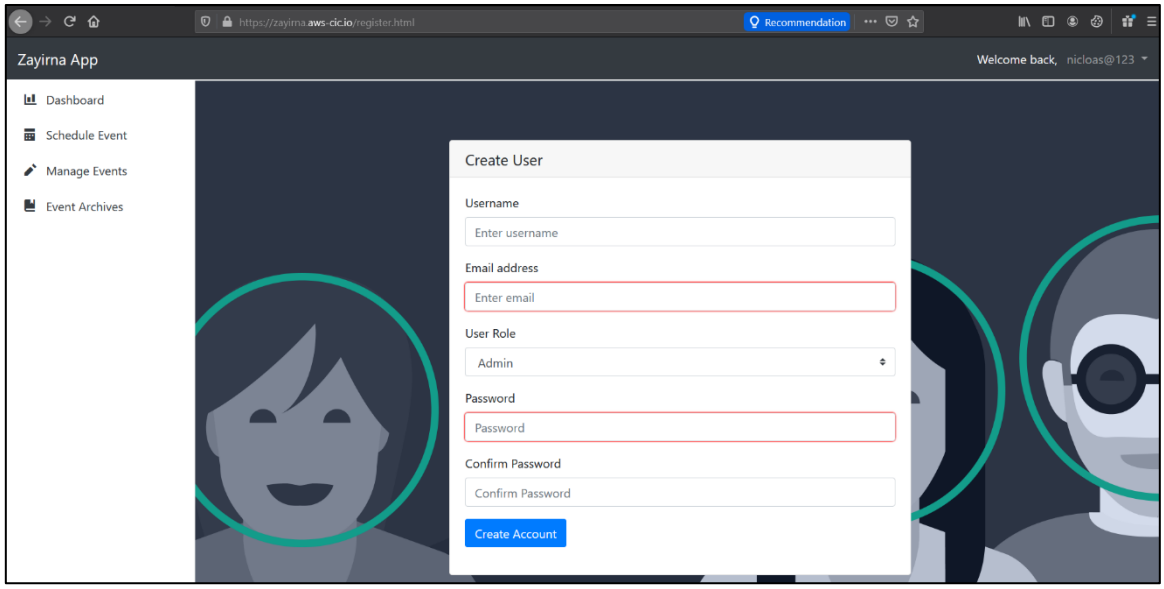


Figure 53 System & User Manual - Registration Page

Similar to the authentication system, the registration procedure is built with a solid validation handling. The below figure displays the error messages if empty entries were submitted in the form.

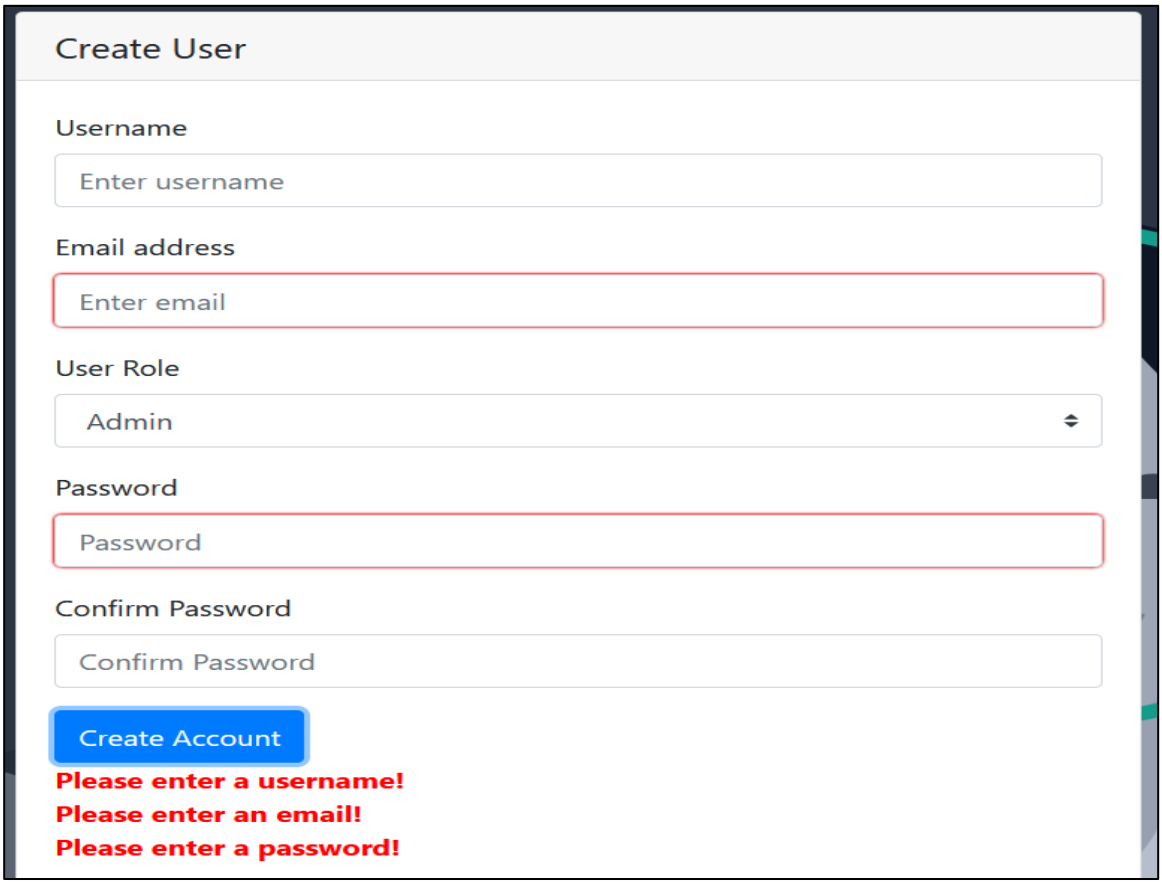


Figure 54 System & User Manual - Registration Missing Entries

If the admin entered a username that already occupied by another existing user, a message will be displayed to notify the admin.

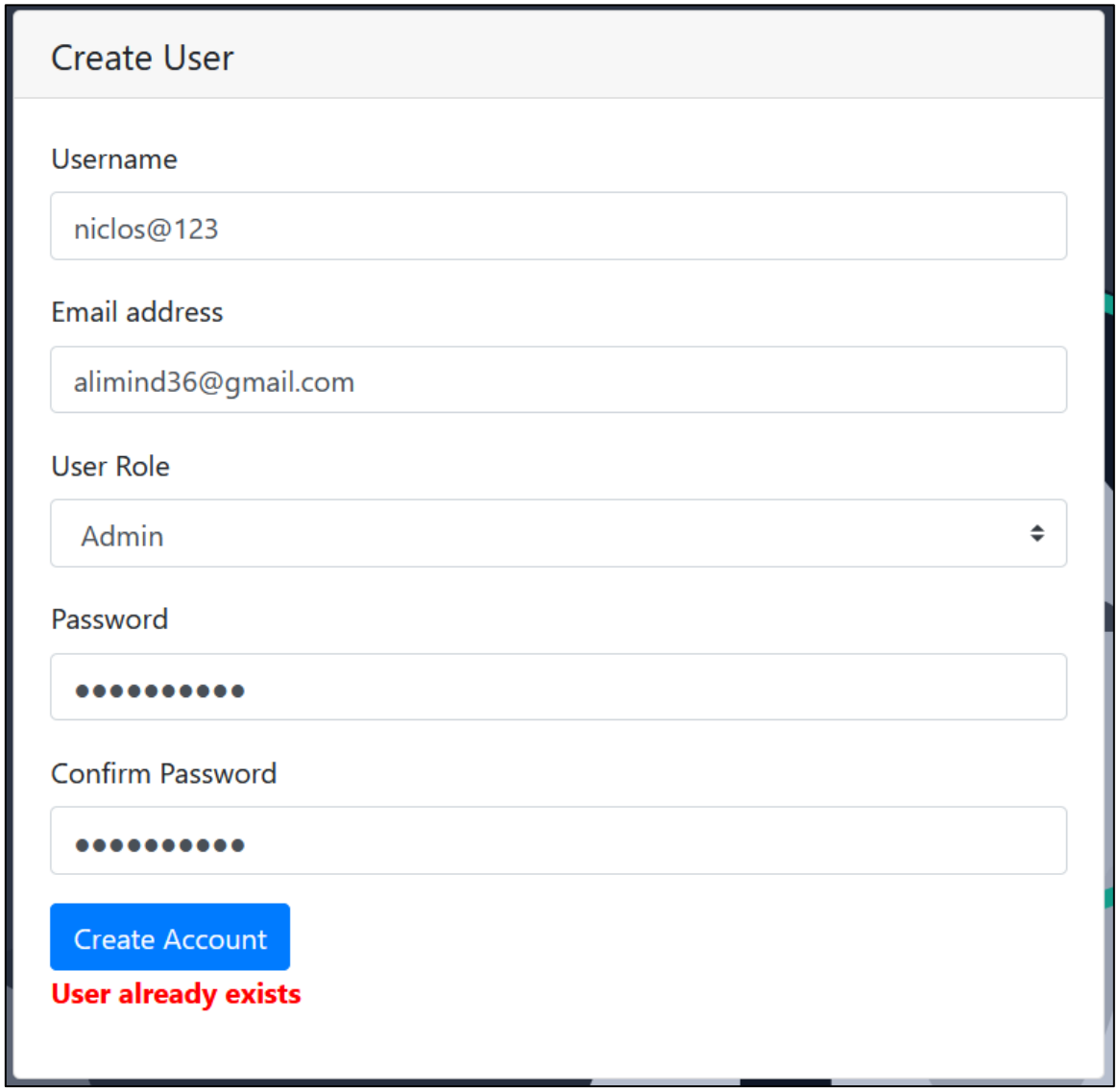


Figure 55 System & User Manual - Existing Username Message

The password has specific criterieas that the admin needs to follow. The password should contain atleast one upper case, lower case, one number and one symbol. Also, the password field needs to match the confirm password field, else an error message will be displayed to the admin as seen in the below figure.

Figure 56 System & User Manual - Mismatching Password

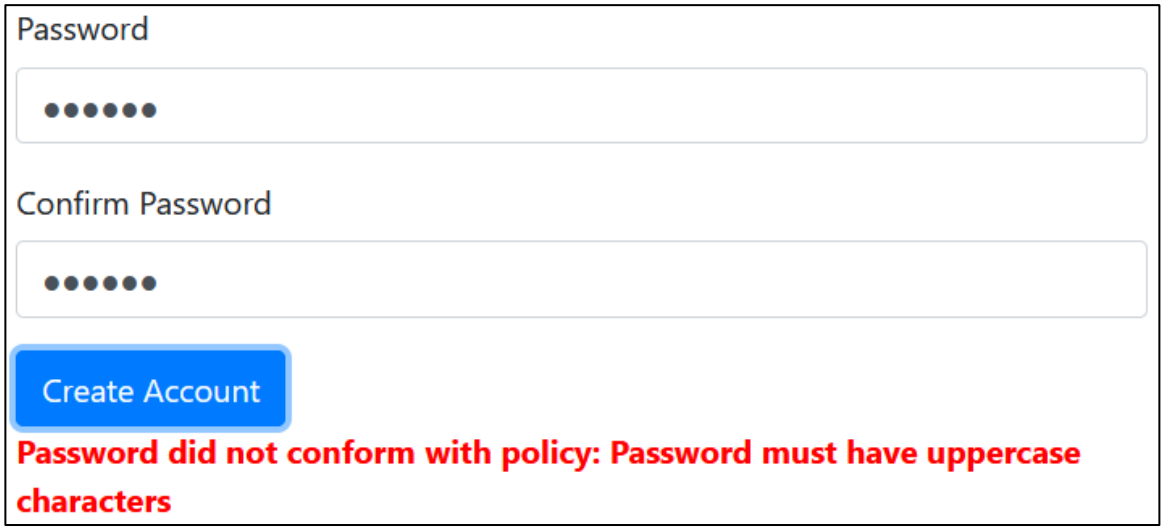


Figure 57 System & User Manual - Missing Uppercase character

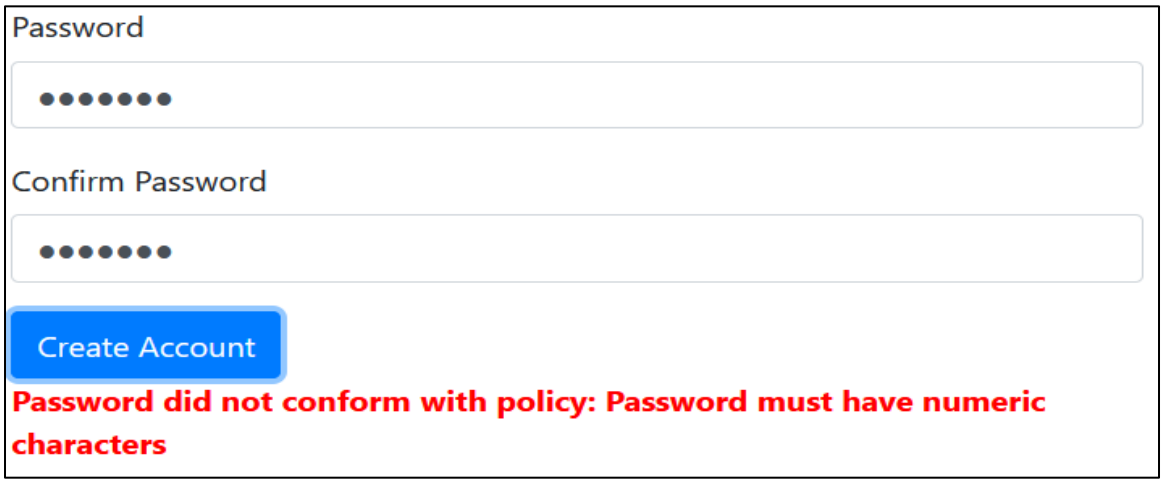


Figure 58 System & User Manual - Missing Numric character

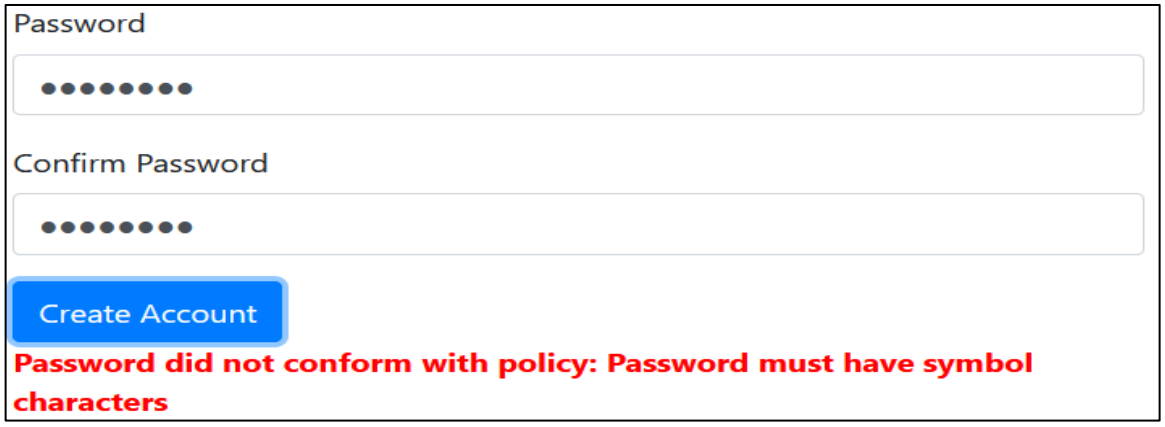


Figure 59 System & User Manual - Missing Symbol character

Once all entries are valid, the system displays a success message to the admin notifying that a verification email is sent to the entered email above.

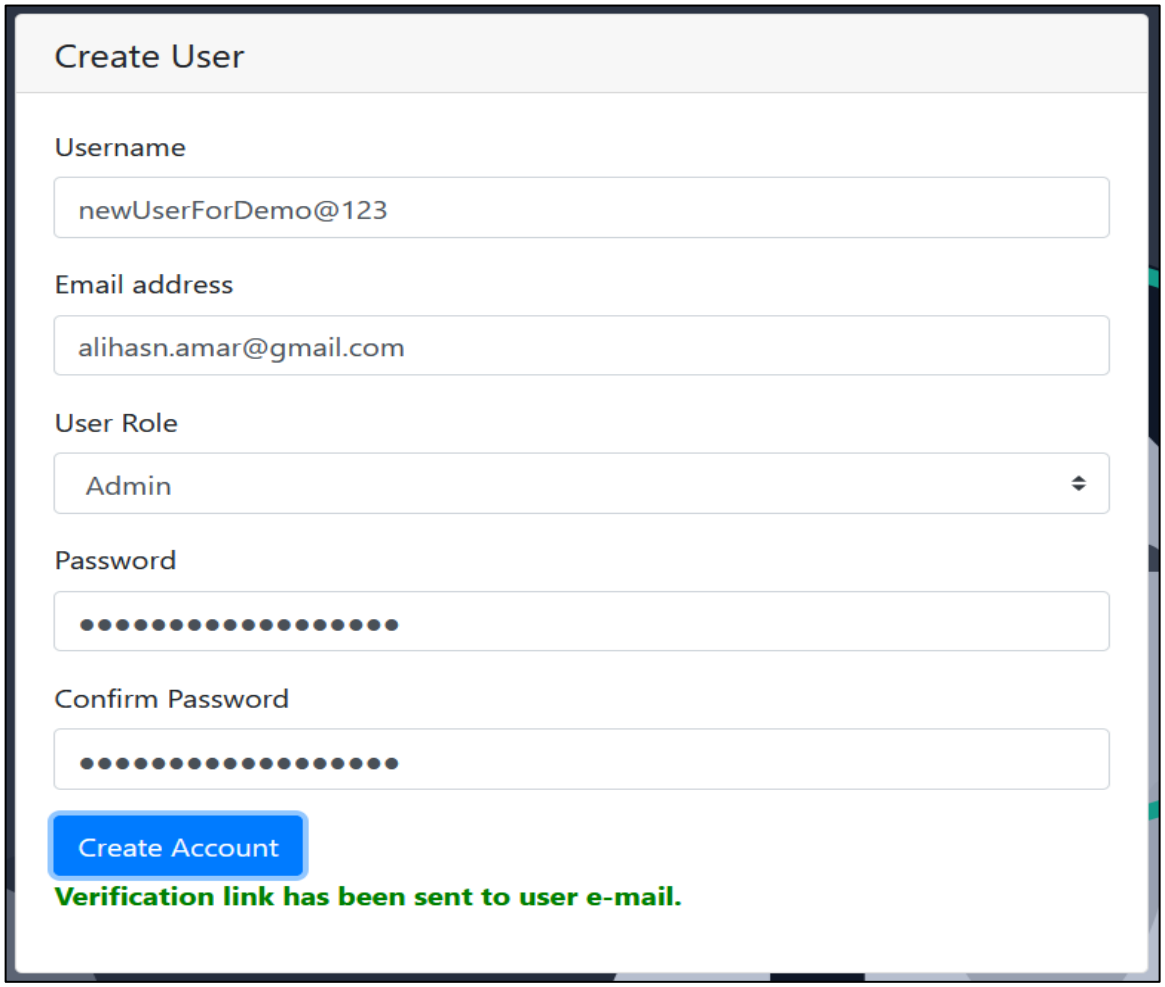


Figure 60 System & User Manual - Verfication Email Message

II. Appendix II: Detailed Design

This section of the appendix contains a detailed design on the system functionalities. Furthermore, this section includes different design diagrams (such as architecture, activity diagrams).

a) Architecture Diagram

❖ Deeplens Heartbeat Section

The below diagram presents the architecture behind the Deeplens Camera Heartbeat subsystem. This subsystem is responsible for rendering the last time the Deeplens has operated. This approach is beneficial in troubleshooting and identifying which camera is not responding. It can be seen from the below image that there are two environments, the first is the Cloud environment. The inserted data of Events DynamoDB table triggers a lambda function that retrieves the last records inserted based on an individual camera. Once the camera has been identified, the lambda function stores the last occurrence time inside another table called Deeplens DynamoDB. This table contains records of each camera along with their last processing time. The second environment is the Local Environment (Client PC), once the client has a session on the web app, a Web Socket API request will be sent (WebSocket API is a work of my colleagues, therefore it will not be discussed in detail). This API is responsible for retrieving the data from the Deeplens DynamoDB and render the content on the web app in real-time.

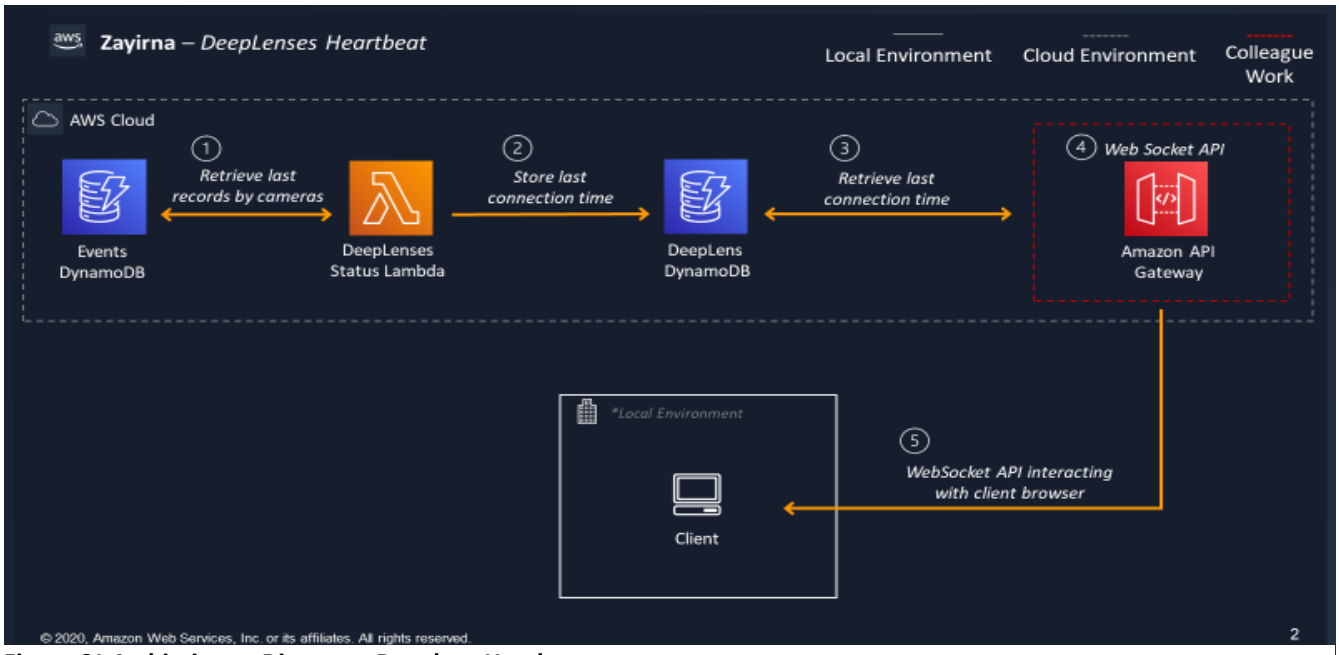


Figure 61 Architeicture Diagram - Deeplens Hearbeat

❖ User Authentication Section

The below diagram represents the authentication subsystem of the web app. This subsystem is the most critical task of the application as it grants access and authorize users to perform specific tasks. AWS provides Cognito services which are responsible for users and group management. This service communicates with the web application through an API (SDK), once a user attempt to log in. The user entries (in Local environments) are sent to Cognito service (in Cloud environment) to get validated.

Cognito response to the application with the proper message depending on the validation outcome.
Furthermore, another scenario of this subsystem is when an Admin registers a new user, once the data entries have been validated, the application sends the user data to Cognito services to create a new record. Cognito then sends a request for Amazon SES to send a verification email to the new user. Once the new user verifies his/her account, they can log in to the web app.

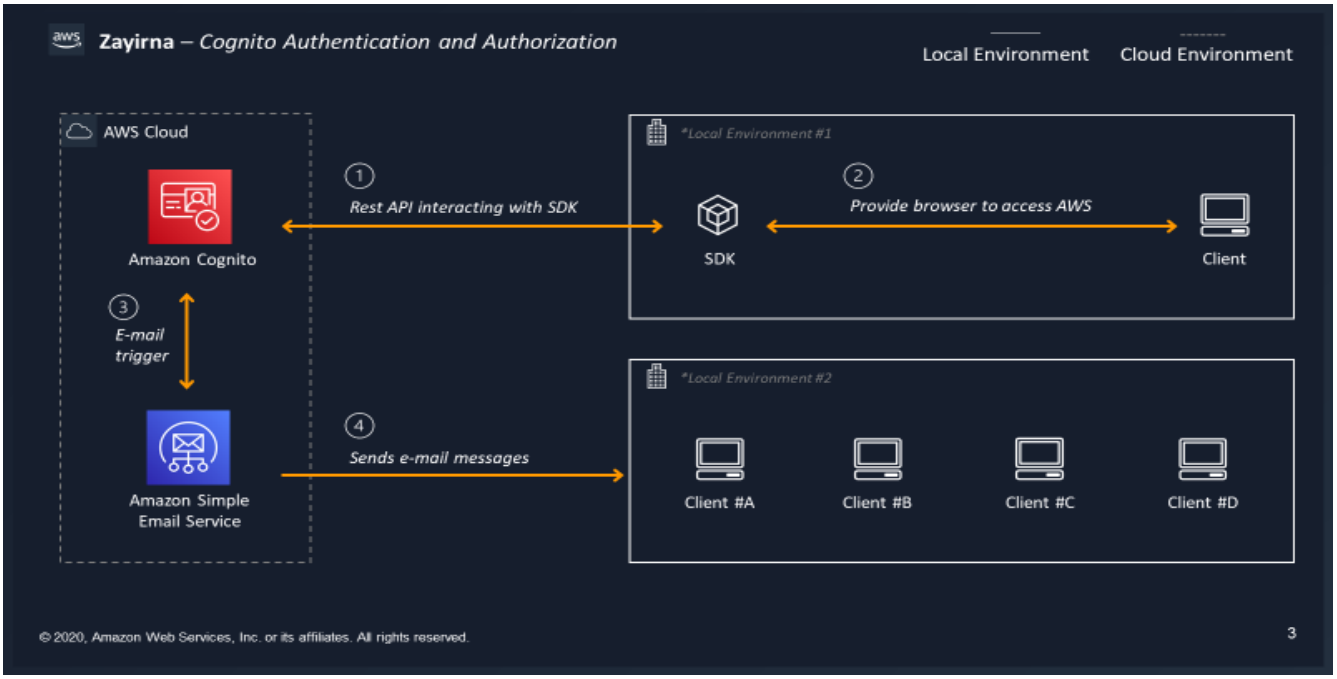


Figure 62 Architeicture Diagram - Cognito Authentication & Autherization

❖ ***Emotion Statistics Section***

According to Hindman B. (2017), attendees may not remember what happened in events but will remember how they felt. Therefore, attendees' satisfaction correlates with their emotions as well as event success. Moreover, having statistics on visitors' emotions is an excellent approach for business owners, especially event organizers, to increase the success of the event. The below figure represents the architecture behind generating the emotion statistics for an individual event.

Whenever a new insertion of face record takes place in Event DynamoDB table, a lambda function (Store Emotion Statistics Lambda) is triggered. This lambda function is responsible for extracting the face emotion from the records and increment the emotion statistics (based on gender, age group and overall emotion group) and updating the Emotion Statistic DynamoDB table. Furthermore, when an event stopped, Store Event Emotion Statistic Lambda is triggered, this lambda involves retrieving all the statistic records from the Emotion Statistic table and convert them into a JSON file. This file then is stored in an S3 bucket (Zayirna) inside Visitor-Emotions folder. Once a user request/opens the emotion statistic page for a specific past event, an API is called. This RESTful API is responsible for calling a lambda function that handles the request (API

Handler Lambda). This lambda opens the JSON file and renders the content of that respective event/exhibition on the browser. It is worth to mention the API is called by an SDK, due to the reason that the web app is hosted in a non-scripting server environment.

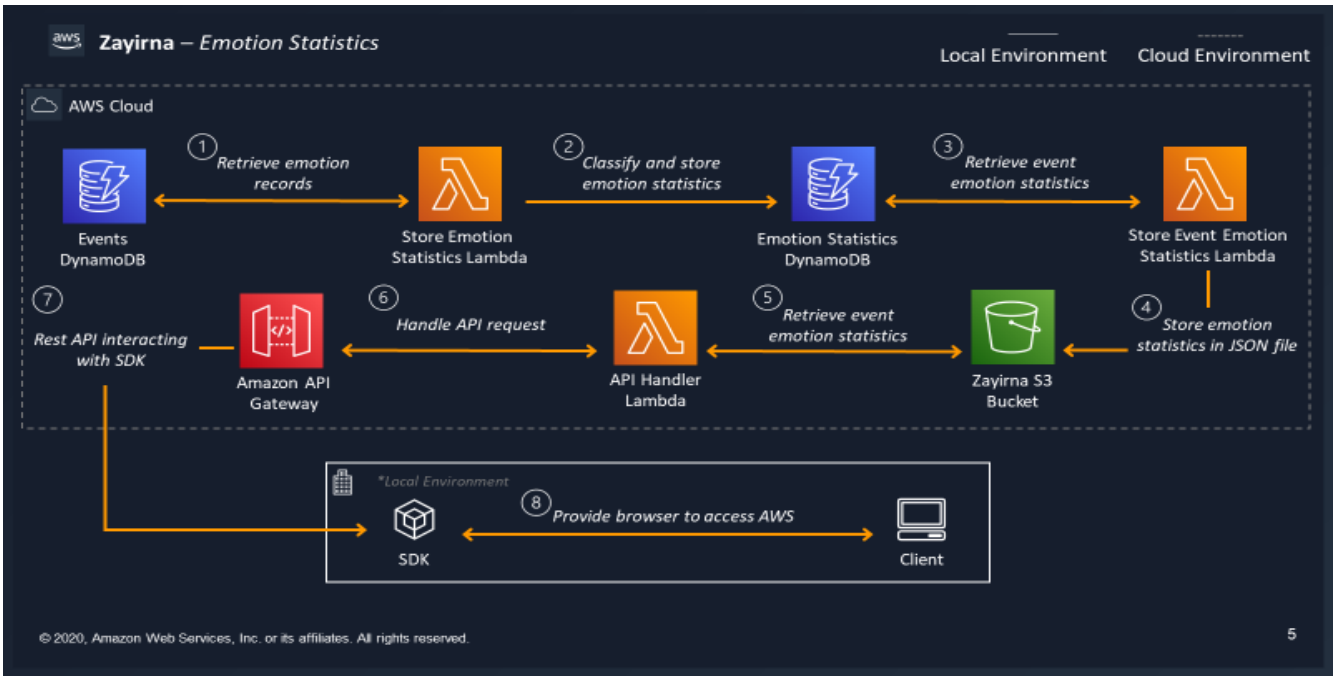


Figure 63 Architeicture Diagram - Emotion Statistics

b) Activity Diagram

❖ *User Authentication Section*

The below diagram displays the flow of action of the authentication process. The activity is initiated as the user request to be sent the login page. The system responds by displaying the login form. Once the user's entries been submitted, the system validates the username and the password. If the entries were valid, the user gets granted to enter the dashboard, and the systems render the page. Whereas if the user entries were invalid, a proper message is displayed, and he/she has three attempts in total. If the number of attempts exceeded three times, an appropriate message is displayed, and the user has to wait 15 minutes until for other attempts.

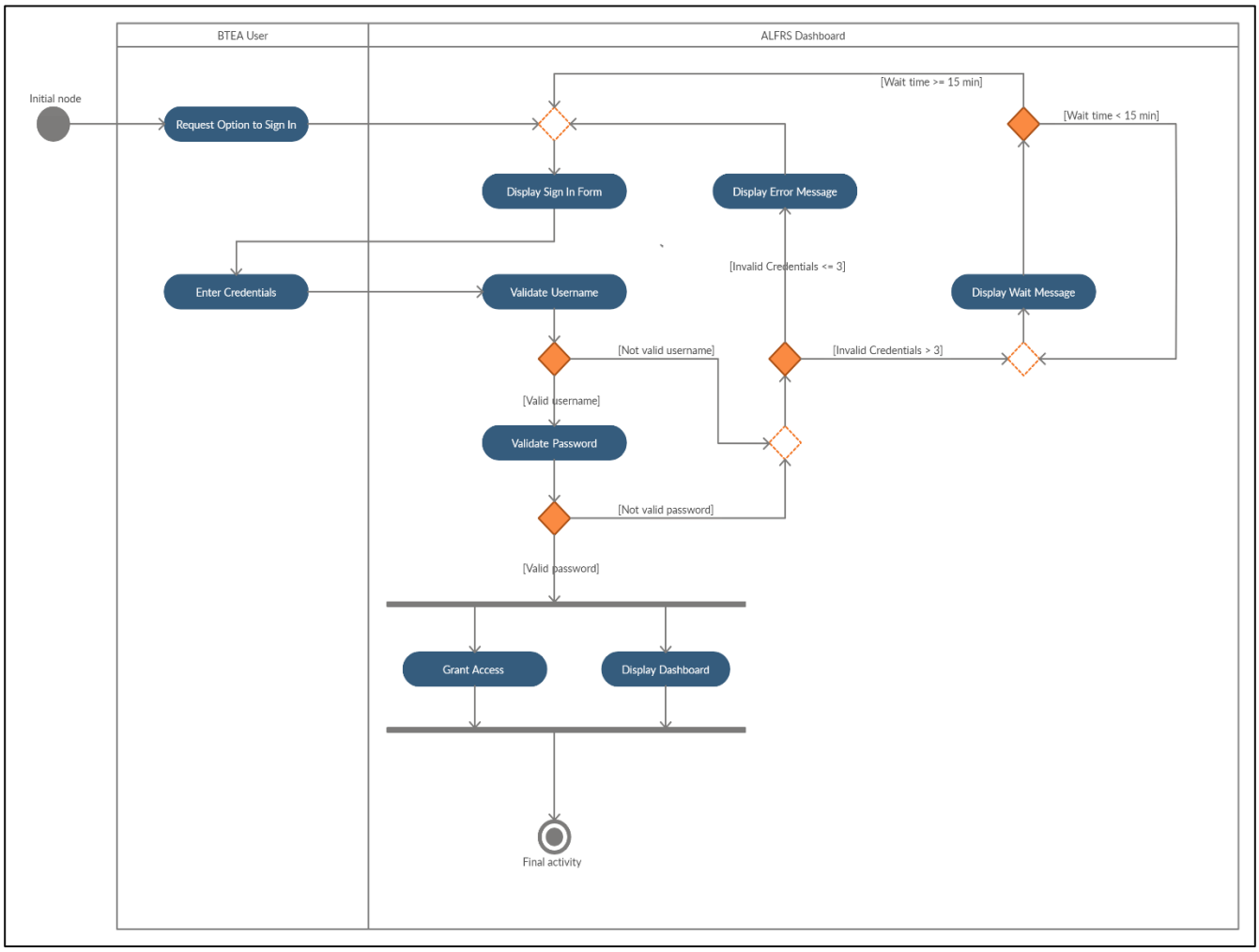


Figure 64 Activity Diagram - User Authentication

❖ ***Emotion Statistics Section***

The below diagram displays the running actions behind the Count Emotion Statistics use-case. As seen from the below figure, it initiates once a facial record is added in the data repository (Events DynamoDB). Firstly, it reads the data and identifies the gender, age group and the captured emotion. Following the identification comes the incremental process. This process contains statements that differentiate the visitor using the previously identified information. Once an individual has been analysed, the statistics are being incremented in the emotion statistic repository (EmotionStatistic DynamoDB). This process iterates with every new face record until the event has stopped. When an event stops, a lambda function (Event Emotion Statistics) retrieves the records of the emotions repository to convert them to JSON and store them in S3 bucket.

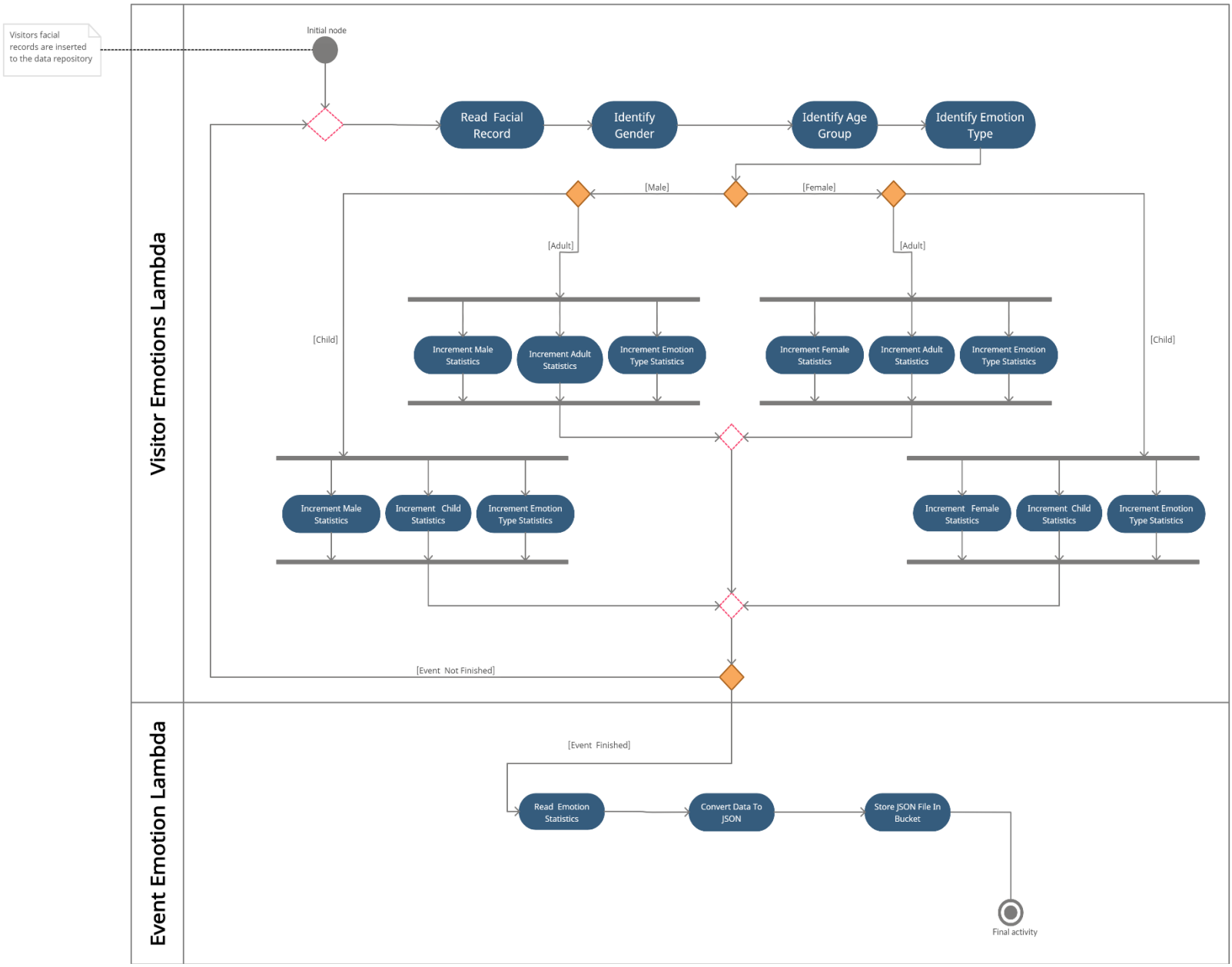


Figure 65 Activity Diagram - Emotion Statistics

III. Appendix III: Detailed Implementation

This section of the appendix contains a detailed description of the permissions and the programmed code behind the system functionalities. Furthermore, this section includes the face detection and recognition process, user authentication and registration, emotion statistics generation and lastly the deeplens heartbeat.

1. Face Detection & Recognition

- Permissions

The inference Lambda has permissions on several services, and these services are AWS Rekognition, AWS S3 Bucket, AWS DynamoDB and AWS Lambda. The following figure displays the policies that are needed for the “deeplens-face-detection” lambda.

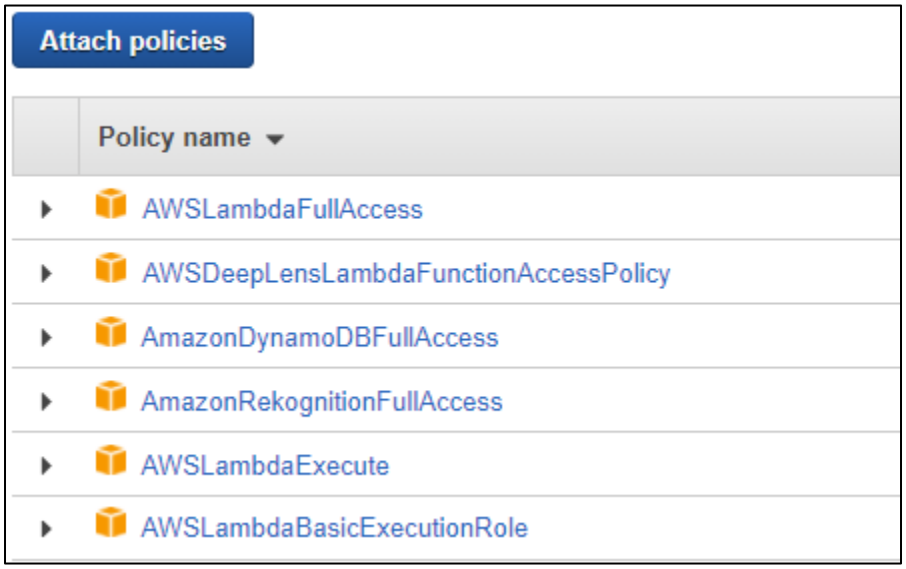


Figure 66 Implementation - Face Detection & Recognition (Permissions – Inference lambda)

Whereas the “store-visitor-data” lambda function has different permission, for instance, the “DeepLens-Access-Encrypted-Bucket” authorized the lambda to read (decrypt) the encrypted files that are stored in the S3 Bucket.

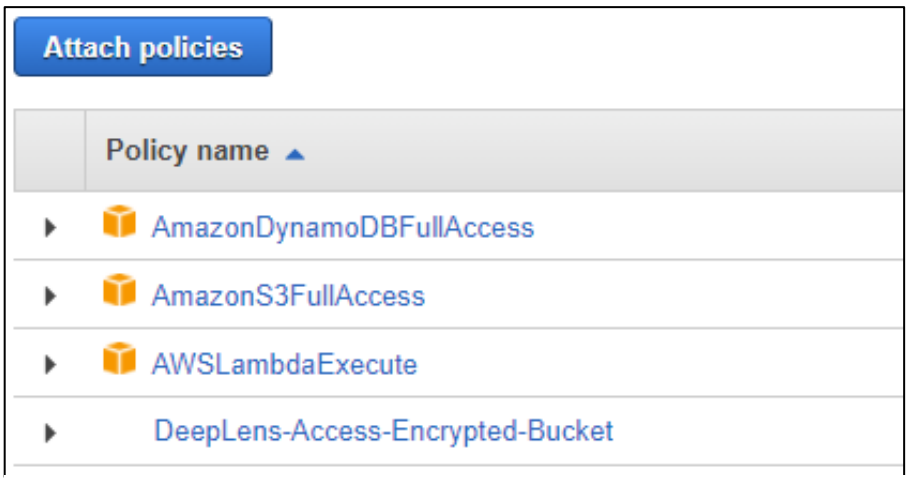


Figure 67 Implementation - Face Detection & Recognition (Permissions – Store Visitor Face)

- Code/Programming

Inference (deeplens-face-detection) Lambda:

An overview on the actions that takes place in this Lambda can be seen from the below diagram.

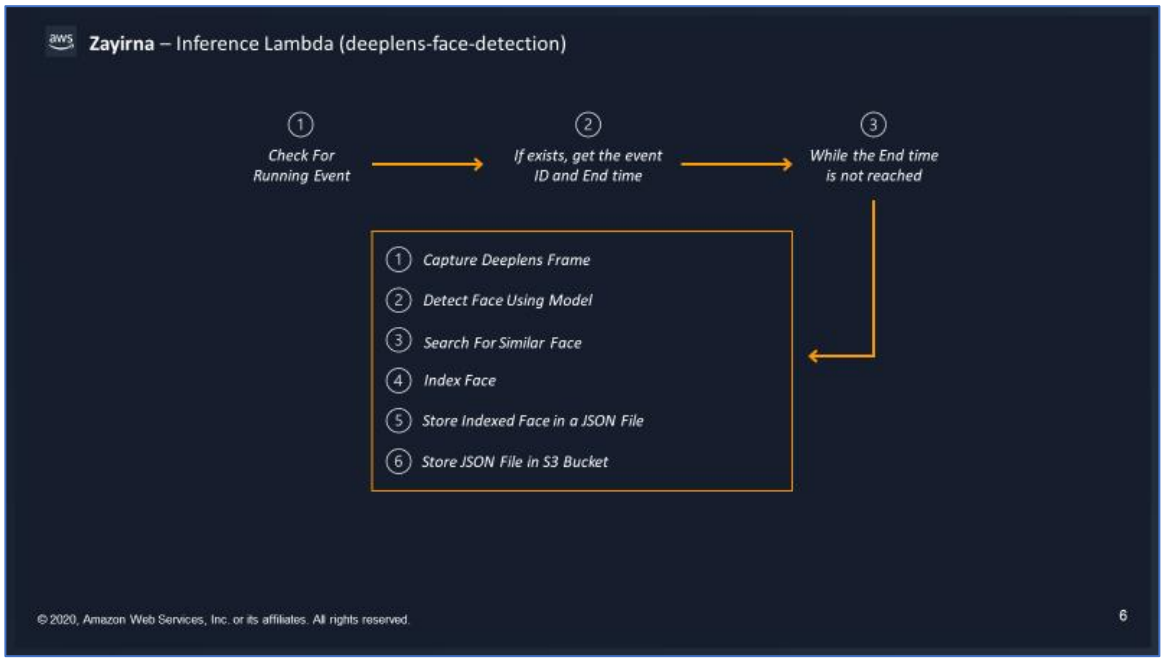


Figure 68 Inference (deeplens-face-detection) Lambda Insight

The first step is to import the libraries, declare and initialized the global variables that will be used throughout the whole python file. As seen from the below figure, variables include table names, camera name and session objects to interact with other AWS services.

```
import json
import awscam
import mo
import cv2
import greengrasssdk
import os
import boto3
import uuid
import datetime
import time
from local_display import LocalDisplay
from botocore.session import Session

dataTable = 'Faces'                             # The name of the DynamoDB table that stores visitors' attributes
eventTable = 'CurrentEvent'                     # Table to get running event ID
loggingTable = 'logs'                           # The name of the DynamoDB table that stores log files
deepLensName = 'dep1'                           # Deeplens Name
facesCollection = "Visitor-Faces"               # Name of the face collection that the AWS Rekognition use to store facial features
faceMatchThreshold = 70                         # The threshold of similarity considering same person face detection

session = Session()                             # Create a session for AWS Services
s3 = session.create_client('s3')                # Create S3 Bucket Session
dynamodb = session.create_client('dynamodb')    # Create DynamoDB Session
dynamodbeResource = boto3.resource('dynamodb')  # Create DynamoDB Resource Object
rekognition = session.create_client('rekognition')  # Create AWS Rekognition Session
```

Figure 69 Implementation - Face Detection & Recognition (Import, declare and initialized variables)

```
106  def CheckForRunningEvent():
107      table = dynamodbeResource.Table(eventTable)
108      resp = table.scan()
109      items = resp['Items']
110      for eachItem in items:
111          if eachItem.get('status') == 'Running':
112              eventID = eachItem['eventID']
113              if eventID is not None:
114                  timeNow = datetime.datetime.now()
115                  timeNow += datetime.timedelta(hours=3)
116                  dateToday = datetime.datetime.strftime(timeNow, '%Y-%m-%d')
117
118                  endTimeStr = eachItem['endTime']
119                  endTimeStr += '.000001'
120                  endTimeStr = endTimeStr[10:]
121                  endTime = dateToday + endTimeStr
122                  endTime = datetime.datetime.strptime(endTime, '%Y-%m-%d %H:%M:%S.%f')
123
124                  infinite_infer_run(eventID, endTime)
125      # Re-run this ever one second
126      Timer(1,CheckForRunningEvent).start()
127  CheckForRunningEvent()
```

Figure 70 Implementation - Face Detection & Recognition (Main function)

The second step is to create a function (called CheckForRunningEvent) that will check for a running event. Once it finds a running event, it calls another function called infinite_infer_run and pass the event and the duration time variables, to begin the detection and the recognition process. Observing the above figure, this function will iterate after each second it finishes executing its content. Initially, this function gets called once (line 127) when the Deeplens Camera is turned on.

The third step is to create an “infinite_infer_run” function that uses the trained model to detects and recognise faces. The below figure shows the path of the trained model, the height and width of images that the model will use to detect faces. Note the function will enter a loop and will finishes until the event concludes.

```
def infinite_infer_run(eventID, endTime):
    """ Run the DeepLens inference loop frame by frame"""

    # Model Info
    model_type = 'ssd'
    model_path = '/opt/awscam/artifacts/mxnet_deploy_ssd_FP16_FUSED.xml'
    model = awscam.Model(model_path, {'GPU': 1})

    # Face & Image Specs
    detection_threshold = 0.5
    input_height = 300
    input_width = 300

    # Sync time with Bahrain add 8 hours
    while endTime > datetime.datetime.now() + datetime.timedelta(hours=8):
```

Figure 71 Implementation - Face Detection & Recognition (Infinite Function - Model & Setup)

The following figure displays a while loop inside the “infinite_infer_run” Lambda function. This loop is responsible for continuously capture the frames of the Deeplens Camera until the end of the running event. The first action that is performed in this loop is to change the size of the captured frame to 300x300. After the image have been resized, it gets passed on to the model to detect a face. If the sent frames contain a face, the model will return the probability of detected faces along with the dimensions of faces in an object format.

The highest four frame objects returned by the model gets selected; it speeds up the operation. Furthermore, the probability of a face has to be more than 50% in order to pass to the recognition process. The second action performed in this loop is to resize the image using the returned dimensions. This approach will be beneficial in case of multiple faces in a single frame. Once an image is resized to an individual face, it gets converted to base64-encoded bytes image, so the AWS Rekognition service accepts the image.

The third action is to compare existed faces with the resized frame using search_faces_by_image function provided by AWS Rekognition service. This function will respond with one if there is an existed similar face, else it will return 0. This approach lets us identify the unique from a redundant face. Once the redundancy identified, the resized frame will be passed to index_faces function to index all the facial features of the face. The response of the index_faces function will be dumped in JSON file that will be stored in S3 bucket. Note, the name of the file plays a vital role in the future.

```
while endTime > datetime.datetime.now() + datetime.timedelta(hours=8):
    try:
        # Get a frame from the video stream
        ret, frame = awscam.getLastFrame()

        if not ret:
            raise Exception('Failed to get frame from the stream')

        frame_resize = cv2.resize(frame, (input_height, input_width))
        parsed_inference_results = model.parseResult(model_type, model.doInference(frame_resize))
        yscale = float(frame.shape[0]) / float(input_height)
        xscale = float(frame.shape[1]) / float(input_width)

        # Get highest propabilities
        getHighProp = 4
        for obj in parsed_inference_results[model_type][0:getHighProp]:
            if obj['prob'] > detection_threshold:
                xminF = int( xscale * obj['xmin'] ) + int((obj['xmin'] - input_width/2) + input_width/2)
                yminF = int( yscale * obj['ymin'] )
                xmaxF = int( xscale * obj['xmax'] ) + int((obj['xmax'] - input_width/2) + input_width/2)
                ymaxF = int( yscale * obj['ymax'] )
                faceImage = frame[yminF:ymaxF, xminF:xmaxF]

                _, jpg_data = cv2.imencode('.jpg', faceImage)
                image = {'Bytes': jpg_data.tostring()}
                faces = rekognition.search_faces_by_image(
                    FaceMatchThreshold=faceMatchThreshold,
                    CollectionId=facesCollection,
                    Image=image,
                    MaxFaces=1
                )

                eventID = eventID.replace('-','')
                facesIndex = None
                key = None

                # Visitor face is already registered in the collection
                if len(faces['FaceMatches']) == 1:
                    # Recognise visitor again and set isRedundant value to true
                    isRedundant = 'true'
                    facesIndex = rekognition.index_faces(
                    Image=image,
                    CollectionId=facesCollection,
                    DetectionAttributes=['ALL'])
                    key = 're-face-' + str(eventID)+ '-' + deepLensName + '-' + str(uuid.uuid1()) + '-' + str(uuid.uuid4()) +'.json'
                # Visitor face is NOT registered in the collection
                else:
                    # Recognise visitor and set isRedundant value to false
                    isRedundant = 'false'
                    facesIndex = rekognition.index_faces(
                    Image=image,
                    CollectionId=facesCollection,
                    DetectionAttributes=['ALL'])
                    key = 'uq-face-' + str(eventID) + '-' + deepLensName + '-'  + str(uuid.uuid1()) + '-' + str(uuid.uuid4())  +'.json'

                bucket_name='visitor-faces-bucket-fr'
                result = s3.put_object(Body=json.dumps(facesIndex), Bucket=bucket_name, Key=key)
    except:
        print('There is an error in the While Loop')
```

Figure 72 Implementation - Face Detection & Recognition (Infinite Function - While loop)

Store-Visitor-Face Lambda:

An overview on the actions that takes place in this Lambda can be seen from the below diagram.

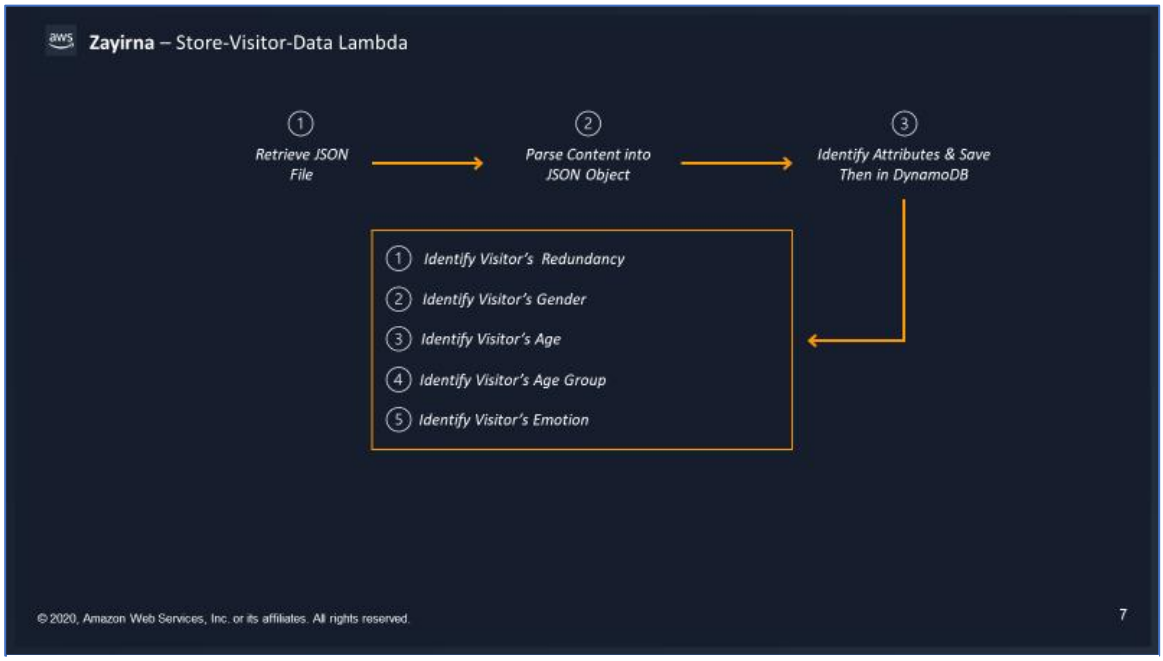


Figure 73 Implementation - Inference Lambda Logic

This function will read the content of the JSON file and determine the face redundancy by reading the prefix as it identifies if the user is redundant or not. Once all the face features are defined, the data will be inserted into a DynamoDB (Events Table).

```
import json
import uuid
import datetime
import os
import time
import boto3

s3 = boto3.client('s3')
dynamodb = boto3.client('dynamodb')
rekognition = boto3.client('rekognition')


dataTable = 'Events'  # The name of the DynamoDB table that stores visitors' attributes
loggingTable = 'logs'  # The name of the DynamoDB table that stores log files
faceMatchThreshold = 70  # The threshold of similarity considering same person face detection
facesCollection = 'Visitor-Faces'  # Name of the face collection that the AWS Rekognition use to store facial features
```

Figure 74 Implementation - Face Detection & Recognition (Store-Visitor-Data Function Libraries & Variables)

```python
def lambda_handler(event, context):
    eventRecord = event['Records'][0]['s3']
    bucket = eventRecord['bucket']['name']
    key = eventRecord['object']['key']

    result = s3.get_object(Bucket=bucket, Key=key)
    faces_string = result["Body"].read().decode()

    faces = json.loads(faces_string)
    faceRecord = faces['FaceRecords'][0]
    faceRecordDetails = faceRecord['FaceDetail']

    isRedundant = None
    if key[:2] == 're':
        isRedundant = 'True'
    else:
        isRedundant = 'False'

    splitKey = key.split('-')
    eventID = splitKey[2]
    deeplens_name = splitKey[3]
    face_id = getVisitorFaceId(faceRecord)
    visitorGender = getVisitorGender(faceRecordDetails)
    low_age_range = getVisitorLowAge(faceRecordDetails)
    high_age_range = getVisitorHighAge(faceRecordDetails)
    meanAge = getVisitorAge(low_age_range, high_age_range)
    ageGroup = getVisitorAgeGroup(meanAge)
    faceEmotion = getVisitorEmotion(faceRecordDetails)

    dynamodb.put_item(
        TableName=dataTable,
        Item={
            'partitionKey': {'S': str(eventID)},
            'metadata': {'S': 'fd-' + str(uuid.uuid4())},
            'faceID': {'S': face_id},
            'ageGroup': {'S': ageGroup},
            'gender': {'S': visitorGender},
            'isRedundant': {'S': str(isRedundant)},
            'emotion': {'S': str(faceEmotion)},
            # 'emotionConfidence': {'S': str(faceEmotionConfidence)},
            'age': {'S': str(int(meanAge))},
            # 'lowAge': {'S': str(low_age_range)},
            # 'highAge': {'S': str(high_age_range)},
            'occurranceTime': {'S': str(datetime.datetime.now() + datetime.timedelta(hours=3))},
            'deeplensName' : {'S': str(deeplens_name)}
        })
```

Figure 75 Implementation - Face Detection & Recognition (Store-Visitor-Data Function Lambda Handler)

2. *User Authentication & Registration*

- Permissions
  No permissions are needed by Amazon Cognito Service

- Code/Programming
  The application communicates with Cognito service via SDK provided by AWS. An app.js folder is created to contain all the needed functions for user authentication and authorization. The below figure presents the available functions inside the file. Initially, Cognito object needs to be declared along with the pool data to use throughout the file; this will minimize code repetition.

Figure 76 Implementation - User Authentication (App.js Content)

The registerButton function gets called once the admin clicks on “create user” button in the registration page. This function is responsible for validating the admin entries, creating Cognito attribute objects from the filled form and signing up the new user. Proper messages will be displayed within a div to identify the admin on the status of the operation. The below image displays how the signup operation acts from the backend point of view.

```
function registerButton(){
    // get user enteries and store them in their respective variables
    var username;
    var useremail;
    var userrole = document.getElementById("roleInputRegister").value;
    var password;

    var errorMes = []

    if( document.getElementById("usernameInputRegister").value){
        username = document.getElementById("usernameInputRegister").value;
    }else{
        errorMes.push('Please enter a username!<br>');
    }


    if(document.getElementById("emailInputRegister").value){
        var useremail = document.getElementById("emailInputRegister").value;
    }else{
        errorMes.push('Please enter an email!<br>');
    }


    if (document.getElementById("passwordInputRegister").value && document.getElementById("confirmationpassword").value) {

        if (document.getElementById("passwordInputRegister").value != document.getElementById("confirmationpassword").value) {
            errorMes.push("Passwords Do Not Match!");
        } else {
            password =  document.getElementById("passwordInputRegister").value;
        }

    } else {
        errorMes.push("Please enter a password!");
    }



    if (errorMes.length != 0){
        $("#spanRegForm").html(errorMes);

    }else{
        // create user pool and store user data in a variable
        var userPool = new CognitoUserPool(poolData);

        // declare a array that will hold attributes of user to record a new cognito account
        var attributeList = [];

        // store the attribute name and value to be create an email identity
        var dataEmail = {
                Name : 'email',
                Value : useremail
        };

        var dataRole = {
            Name : 'custom:role',
            Value : userrole
        };

        // create an email identity for the new user
        var attributeEmail = new AmazonCognitoIdentity.CognitoUserAttribute(dataEmail);

        // create an role identity for the new user
        var attributeRole = new AmazonCognitoIdentity.CognitoUserAttribute(dataRole);

        // store all identities for the new user into the attribute array
        attributeList.push(attributeEmail);
        attributeList.push(attributeRole);

        // create a record for the new user to be registered in aws cognito service
        userPool.signUp(username, password, attributeList, null, function(err, result){
                if (err) {
                    if(err.message != 'Unkown error' || JSON.stringify(err) == 'Unkown error' ){
                        // alert(err.message || JSON.stringify(err));
                        $("#spanRegForm").html('');
                        $("#spanRegFormSucc").html('');
                        $("#spanRegForm").html(err.message || JSON.stringify(err));
                        return;
                    }
                }
                $("#spanRegForm").html('');
                cognitoUser = result.user;
                console.log('user name is ' + cognitoUser.getUsername());

                // display notification message to go check email to confirm registration
                $("#spanRegFormSucc").html("Verification link has been sent to user e-mail.");
        });
        }



}
```

Figure 77 Implementation - User Authentication (Register User Function)

The signInButton function is called once a user attempt to login into the application. This function involves invalidating the form entries, creating authentication and Cognito objects as well as limiting the number of failed attempts to the user; the user has maximum three attempts to log in, in case of failure in authentication for all three attempts, the user will have to wait 15 minutes.

```
function signInButton(){
    // get user credentails and store in variable
    if(document.getElementById("inputUsername").value && document.getElementById("inputPassword").value){
        var authenticationData = {
        Username : document.getElementById("inputUsername").value,
        Password : document.getElementById("inputPassword").value,
        };

        // create user pool and store user data in a variable
        var userPool = new CognitoUserPool(poolData);
        var userData = {
              Username : document.getElementById("inputUsername").value,
              Pool : userPool,
        };

        // create cognito authentication object from entered user credentials
        var authenticationDetails = new AuthenticationDetails(authenticationData);

        // create a cognito user object
        var cognitoUser = new CognitoUser(userData);

        // Check if attempt times is running
        if(localStorage.getItem("attemptTimeout") !== null ){

            var gDate = new Date();
            var gTime = gDate.getTime();
            if(parseInt(gTime) >= parseInt(localStorage.getItem("attemptTimeout")) ){ ...
            }else{ ...

            }
        }else{
            // authenticate the cognito user by using cognito authentication object
            cognitoUser.authenticateUser(authenticationDetails, {
                onSuccess: function (result) {
                    $("#spanLogFormSucc").html('');
                    var accessToken = result.getAccessToken().getJwtToken();
                    console.log(accessToken);
                    cognitoUser.getUserAttributes(function(err, result) {
                        localStorage.setItem("authUserRole",result[1].getValue());
                        console.log(result[1].getValue());
                    });
                    setTimeout(function(){ window.location.href = "dashboard.html"; }, 500);
                    // localStorage.removeItem("attemptCounter")

                },
                onFailure: function(err) {
                    var attemptCounter;
                    if(localStorage.getItem("attemptCounter") === null){
                        attemptCounter = localStorage.setItem("attemptCounter",1);
                        $("#spanLogFormSucc").html('');
                        $("#spanLogFormSucc").html(err.message || JSON.stringify(err));
                    }else{
                        if(parseInt(localStorage.getItem("attemptCounter")) === 3){
                            // Have excceded number of attempts
                            $("#spanLogFormSucc").html('');
                            $("#spanLogFormSucc").html('Please wait 15 minutes to try again!');
```

Figure 78 Implementation - User Authentication (Login User Function 1)

```
                                // Save when user is able to re-attempt
                                var tDate = new Date();
                                var tTime = tDate.getTime();
                                var schueledTime = parseInt(tTime)+900000; //900000 Wait 15 Minutes
                                localStorage.setItem("attemptTimeout", schueledTime)

                            }else if(parseInt(localStorage.getItem("attemptCounter")) === 2){
                                incrementCounter = parseInt(localStorage.getItem("attemptCounter")) + 1;
                                attemptCounter = localStorage.setItem("attemptCounter",incrementCounter);
                                $("#spanLogFormSucc").html('');
                                $("#spanLogFormSucc").html('Last attempt, in fail scenario wait 15 minutes to try again!');

                            }else{
                                // Trully wrong error
                                incrementCounter = parseInt(localStorage.getItem("attemptCounter")) + 1;
                                attemptCounter = localStorage.setItem("attemptCounter",incrementCounter);
                                $("#spanLogFormSucc").html('');
                                $("#spanLogFormSucc").html(err.message || JSON.stringify(err));
                            }
                        }

                    },
                });
            }
        }else{
            var errMs = [];
            if(!document.getElementById("inputUsername").value){
                errMs.push('Please enter a username!<br>');
            }if(!document.getElementById("inputPassword").value){
                errMs.push('Please enter a password!<br>');
            }

            $("#spanLogFormSucc").html(errMs);
        }
    }
```

Figure 79 Implementation - User Authentication (Login User Function 2)

The renderName function involves displaying the name of the user in all the pages once he/she is logged in. The below figure shows how the name of the user is taken and rendered to a specific divs.

```
function renderName() {
    // create user pool and store user data in a variable
    var userPool = new CognitoUserPool(poolData);

    // get current user (from local storage) who is occupating cognito session
    var cognitoUser = userPool.getCurrentUser();

    // render user name on the dashboard page
    if (cognitoUser != null) {
        cognitoUser.getSession(function(err, session) {
            if (err) {
                alert(err);
                return;
            }
            console.log(cognitoUser);
            $('#usernameDashboard').html(cognitoUser.username);
        });
    }
}
```

Figure 80 Implementation - User Authentication (Render Name Function)

VerifyPageAccess function is responsible for checking if the person who attempts to enter the page is a logged-in user or not. In case of non-logged-in visitors, they will be redirected to the login page as seen from the below figure.

```
285  function verifyPageAccess(){
286      // create user pool and store user data in a variable
287      var userPool = new CognitoUserPool(poolData);
288
289      // get current user (from local storage) who is occupating cognito session on
290      var cognitoUser = userPool.getCurrentUser();
291
292      // verify no active user
293      if (cognitoUser == null) {
294          // redirect to welcome page
295          window.location.href = "login.html";
296      }
297  }
```

Figure 81 Implementation - User Authentication (Verify Page Access Function)

VerifyPageAuthorization function involves verifying the logged-in user if they are authorized to enter a specific page or not. Users with the “Admin” role have the top privileged type of users, whereas staff, for instance, cannot enter several pages such as user registry page.

```
299  function verifyPageAuthorization(){
300      // create user pool and store user data in a variable
301      var userPool = new CognitoUserPool(poolData);
302
303      // get current user (from local storage) who is occupating cognito session
304      var cognitoUser = userPool.getCurrentUser();
305
306      // verify if the active user is admin
307      if (cognitoUser = null) {
308          // not a user
309          window.location.href = "login.html";
310      }
311      else{
312          // check user role
313          var userrole = localStorage.getItem("authUserRole");
314          if(userrole != 'admin'){
315              window.location.href = "dashboard.html";
316          }
317      }
318  }
```

Figure 83 Implementation - User Authentication (Verify Page Authorization Function)

The following figure presents the function that is responsible for displaying the admin tabs or navigations. This function checks the user role to determine whether to display the tabs or not.

```
function renderAdminTabs(){
    // create user pool and store user data in a variable
    var userPool = new CognitoUserPool(poolData);

    // get current user (from local storage) who is occupating cognito session
    var cognitoUser = userPool.getCurrentUser();

    // verify if the active user is admin
    if (cognitoUser != null) {
        // check user role
        var userrole = localStorage.getItem("authUserRole");
        if(userrole == 'admin'){
            $("#dropdownlist").prepend("<a class='dropdown-item' href='register.html'><span class='oi oi-person'></span>  Create Users</a>");
            $("#dropdownlist2").prepend("<a class='dropdown-item' href='register.html'><span class='oi oi-person'></span>  Create Users</a>");
        }
    }
}
```

Figure 84 Implementation - User Authentication (Render Admin Tabs Function)

3. *Calculating Emotion Statistics*

- Permissions
  The "Store-Emotion-Statistics" function has two policies that allow to execute a lambda function and read and write access to DynamoDB table.

Figure 85 Implementation - Emotion Statistics (Permissions – Store Emotion Statistics)

The "Store-Event-Emotion-Statistics" function has some additional policy in comparing to the previous function. One permission provides write access to the S3 Bucket, and another permission is to call recognition service.

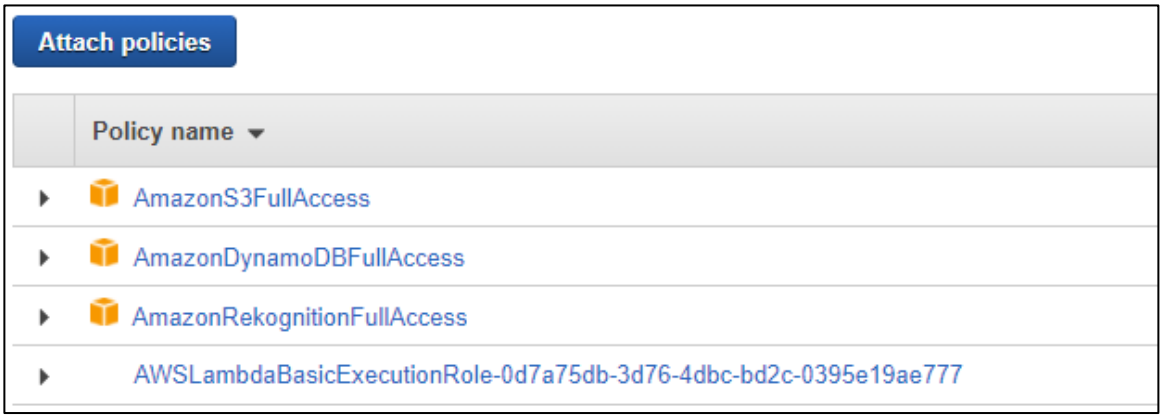


Figure 86 Implementation - Emotion Statistics (Permissions – Store Event Emotion Statistics)

The "VisitorEmotionsAPI" function has the standard permission for a lambda function; that includes S3 bucket access and Lambda executing permission.

Figure 87 Implementation - Emotion Statistics (Permissions – Visitor Emotion API)

- Code/Programming

  In case of a new face record added to the Events DynamoDB Table, a lambda function is responsible for retrieving the record and read the data to increment the statistics in the Emotion-Statistics Table. The below figure represents the python lambda function that performs the above operation.

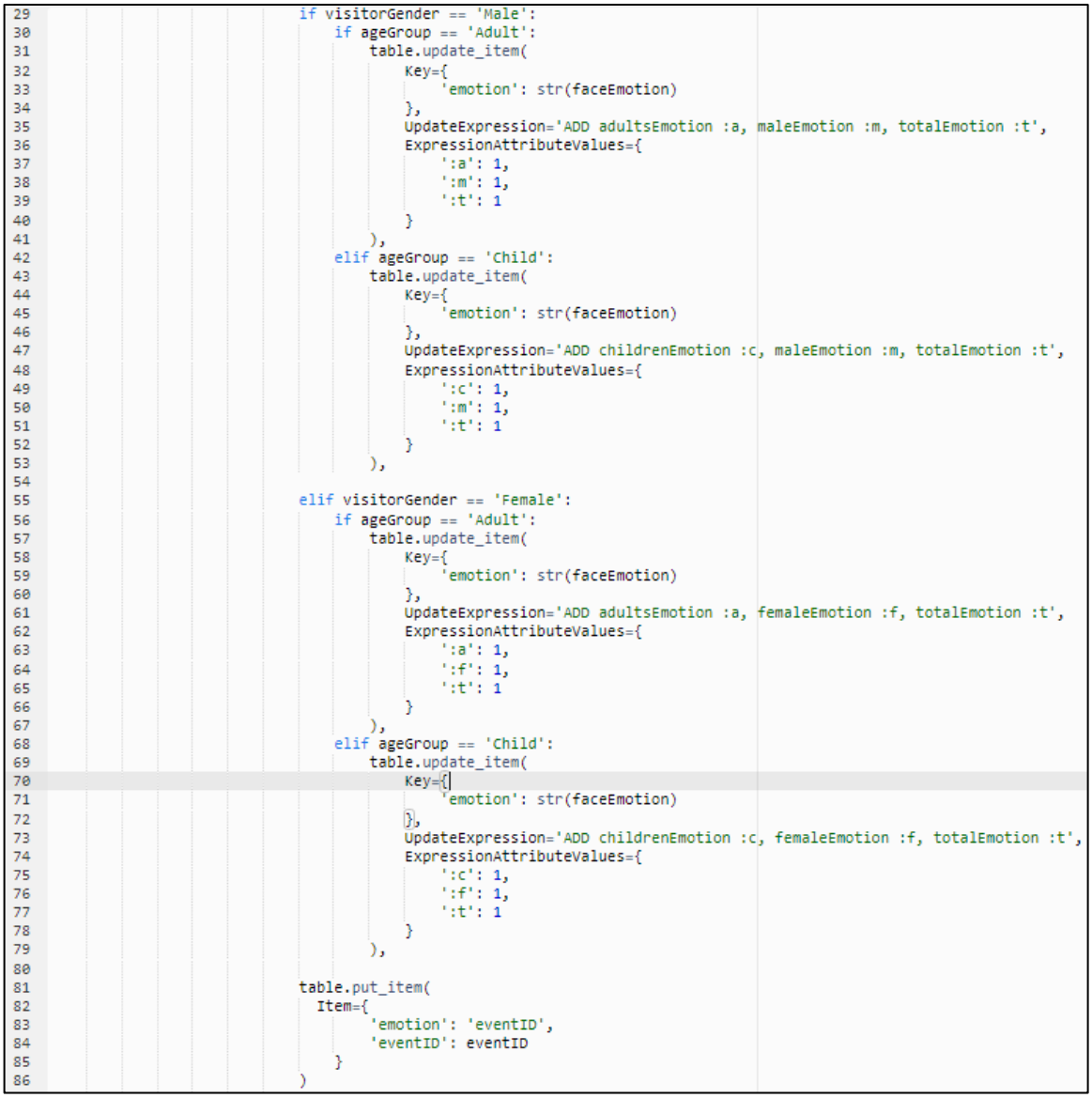
  Store -Emotion-Statistics Function

```python
import json
import boto3
import datetime
from boto3.dynamodb.conditions import Key

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('Emotion-Statistics')

dynamodbtest = boto3.client('dynamodb')
test_table = 'logs'

def lambda_handler(event, context):

    for eachRec in event['Records']:

        if eachRec['eventName'] == "INSERT":

            if 'gender' in eachRec.get('dynamodb').get('NewImage'):
                if 'emotion' in eachRec.get('dynamodb').get('NewImage'):
                    if 'age' in eachRec.get('dynamodb').get('NewImage'):
                        if 'ageGroup' in eachRec.get('dynamodb').get('NewImage'):
                            eventID = eachRec.get('dynamodb').get('NewImage').get('partitionKey').get('S')
                            visitorGender = eachRec.get('dynamodb').get('NewImage').get('gender').get('S')

                            meanAge = eachRec['dynamodb']['NewImage']['age']['S']
                            ageGroup = eachRec['dynamodb']['NewImage']['ageGroup']['S']
                            faceEmotion = eachRec['dynamodb']['NewImage']['emotion']['S']
```

Figure 88 Implementation - Emotion Statistics (Store Emotion Statistics 1)

As seen from the above figure, after the importing of the needed libraries, the lambda checks if the action of the new event was inserted and not modified. Once it fulfils this condition, it checks for all needed attributes and stores them in variables.

Upon the saved variables, the lambda determines what statistics needs to be updated. Observing the below figure, the lambda categories the process to four categories (Male, Female, Adult, Child), each category represent a specific statistic in the Emotion-Statistics Table.



```
                    if visitorGender == 'Male':
                        if ageGroup == 'Adult':
                            table.update_item(
                                Key={
                                    'emotion': str(faceEmotion)
                                },
                                UpdateExpression='ADD adultsEmotion :a, maleEmotion :m, totalEmotion :t',
                                ExpressionAttributeValues={
                                    ':a': 1,
                                    ':m': 1,
                                    ':t': 1
                                }
                            ),
                        elif ageGroup == 'Child':
                            table.update_item(
                                Key={
                                    'emotion': str(faceEmotion)
                                },
                                UpdateExpression='ADD childrenEmotion :c, maleEmotion :m, totalEmotion :t',
                                ExpressionAttributeValues={
                                    ':c': 1,
                                    ':m': 1,
                                    ':t': 1
                                }
                            ),

                    elif visitorGender == 'Female':
                        if ageGroup == 'Adult':
                            table.update_item(
                                Key={
                                    'emotion': str(faceEmotion)
                                },
                                UpdateExpression='ADD adultsEmotion :a, femaleEmotion :f, totalEmotion :t',
                                ExpressionAttributeValues={
                                    ':a': 1,
                                    ':f': 1,
                                    ':t': 1
                                }
                            ),
                        elif ageGroup == 'Child':
                            table.update_item(
                                Key={
                                    'emotion': str(faceEmotion)
                                },
                                UpdateExpression='ADD childrenEmotion :c, femaleEmotion :f, totalEmotion :t',
                                ExpressionAttributeValues={
                                    ':c': 1,
                                    ':f': 1,
                                    ':t': 1
                                }
                            ),

                    table.put_item(
                      Item={
                            'emotion': 'eventID',
                            'eventID': eventID
                        }
                    )
```

Figure 89 Implementation - Emotion Statistics (Store Emotion Statistics 2)

Store-Event-Emotion-Statistics Function

Once the status of the running event is changed to “Stopped”, a lambda function gets triggered by this modification action, this lambda called Store-Event-Emotion-Statistics. This function is responsible for retrieving all the data from the Emotion-Statistics DynamoDB Table and store them in JSON file inside the web app’s S3 Bucket. The following figure displays the libraries needed for the program to run as well as it displays the code for verifying that the event changed from “Running” to “Stopped”.

```
import json
import boto3
import datetime

loggingTable = 'logs'
facesCollection = "Visitor-Faces"
rekognition = boto3.client('rekognition')
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('Emotion-Statistics')
tableDL = dynamodb.Table('DeepLens')
s3 = boto3.client('s3')

def lambda_handler(event, context):

    for eachRec in event['Records']:

        if 'eventName' in eachRec and eachRec['eventName'] == "MODIFY":

            if 'status' in eachRec['dynamodb']['NewImage']:
                if 'status' in eachRec['dynamodb']['OldImage']:
                    newEvent = eachRec['dynamodb']['NewImage']['status']['S']
                    oldEvent = eachRec['dynamodb']['OldImage']['status']['S']

                    if oldEvent == "Running" and newEvent == "Stopped":


                        emotions = ["SAD", "CALM", "ANGRY", "HAPPY", "CONFUSED", "FEAR", "DISGUSTED", "SURPRISED"]
                        statType = ["adultsEmotion", "childrenEmotion", "maleEmotion", "femaleEmotion", "totalEmotion"]

                        eventID = (table.get_item(Key={'emotion': 'eventID'}))['Item']['eventID']

                        totalEmotion =[]
                        maleEmotion = []
                        femaleEmotion = []
                        adultsEmotion = []
                        childrenEmotion =[]
```

Figure 90 Implementation - Emotion Statistics (Store Event Emotion Statistics 1)

After identifying the variables, the lambda iterates through Emotion-Statistics DynamoDB Table column by column to conclude with arrays containing adults, children, male and female emotions as well as the overall emotion type. Once the iteration finishes, the statistics are inserted into a JSON format variable followed by storing this variable in JSON file inside the web app S3 Bucket.

As seen from the below figure, the lambda truncates the content of the Deeplens Status table and reset all records in Emotion-Statistics table to 0. Finally, it deletes and recreates all the face index collections preparing for the next event.

```
                    for eachStatType in statType:
                        for eachEmotion in emotions:
                            data = table.get_item(Key={'emotion': eachEmotion})
                            if data['Item'].get(eachStatType) is not None:
                                data = int(data['Item'].get(eachStatType))
                                if data is not None:
                                    if eachStatType == "adultsEmotion":
                                        adultsEmotion.append(data)
                                    elif eachStatType == "childrenEmotion":
                                        childrenEmotion.append(data)
                                    elif eachStatType == "maleEmotion":
                                        maleEmotion.append(data)
                                    elif eachStatType == "femaleEmotion":
                                        femaleEmotion.append(data)
                                    elif eachStatType == "totalEmotion":
                                        totalEmotion.append(data)

                    data = {}
                    data['emotion'] = []
                    data['emotion'].append({
                        'eventID' : eventID,
                        'totalEmotions' : totalEmotion,
                        'femaleEmotions' : femaleEmotion,
                        'maleEmotions' : maleEmotion,
                        'adultsEmotions' : adultsEmotion,
                        'childrenEmotions' : childrenEmotion
                    })

                    # Store the whole emotions captured during the EVENT
                    bucket_to='zayirna'
                    key_name = 'visitorEmotions/' + eventID + '.json'
                    result = s3.put_object(Body=json.dumps(data), Bucket=bucket_to, Key=key_name)

                    # Clear DeepLens Table
                    deepLensList = tableDL.scan()
                    for eachItem in deepLensList['Items']:
                        response = tableDL.delete_item(
                            Key={
                                'deepLensName': eachItem['deepLensName']
                            }
                        )

                    # Clear Emotion Statistics Table
                    with table.batch_writer() as batch:
                        for emotion in emotions:
                            batch.put_item(
                                Item={
                                     "adultsEmotion": 0,
                                     "childrenEmotion": 0,
                                      "emotion": emotion,
                                      "femaleEmotion": 0,
                                      "maleEmotion": 0,
                                      "totalEmotion": 0
                                }
                            )
                    print('Tables are been Cleared')

                    rekognition.delete_collection(CollectionId=facesCollection)
                    rekognition.create_collection(CollectionId=facesCollection)
```

Figure 91 Implementation - Emotion Statistics (Store Event Emotion Statistics 2)

RESTful API Handler Lambda

This function is responsible for reading the JSON files of all events' statistics (emotion) and store them in an array. This array will be sent in the body of the API response. The following figure demonstrates how the lambda function handles the requests.

```python
import json
import boto3

s3 = boto3.client('s3')
bucketname = 'zayirna'

def lambda_handler(event, context):


    if event['httpMethod'] == 'GET':
        listItems = s3.list_objects_v2(
            Bucket = bucketname,
            Prefix = 'visitorEmotions/'
            )

        emotions = []
        arrayList = listItems['Contents']
        for eachItem in arrayList:
            result = s3.get_object(Bucket=bucketname, Key=eachItem['Key'])
            emotion_string = result["Body"].read().decode()
            emotion = json.loads(emotion_string)
            emotions.append(emotion)

        return {
            'statusCode': 200,
            'headers': {
                'Access-Control-Allow-Headers': 'Content-Type',
                'Access-Control-Allow-Origin': '*',
                'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
            },
            'body': json.dumps(emotions, indent=4, sort_keys=True, default=str)
        }

```

Figure 92 Implementation - Emotion Statistics (RESTful API – API Lambda Handler)

As seen from the below two figure, scripts tags are added to import the files of the downloaded SDKs, and the API client object is created to access the call the API.

```
<script type="text/javascript" src="js/apiGateway-js-sdk/lib/axios/dist/axios.standalone.js"></script>
<script type="text/javascript" src="js/apiGateway-js-sdk/lib/CryptoJS/rollups/hmac-sha256.js"></script>
<script type="text/javascript" src="js/apiGateway-js-sdk/lib/CryptoJS/rollups/sha256.js"></script>
<script type="text/javascript" src="js/apiGateway-js-sdk/lib/CryptoJS/components/hmac.js"></script>
<script type="text/javascript" src="js/apiGateway-js-sdk/lib/CryptoJS/components/enc-base64.js"></script>
<script type="text/javascript" src="js/apiGateway-js-sdk/lib/url-template/url-template.js"></script>
<script type="text/javascript" src="js/apiGateway-js-sdk/lib/apiGatewayCore/sigV4Client.js"></script>
<script type="text/javascript" src="js/apiGateway-js-sdk/lib/apiGatewayCore/apiGatewayClient.js"></script>
<script type="text/javascript" src="js/apiGateway-js-sdk/lib/apiGatewayCore/simpleHttpClient.js"></script>
<script type="text/javascript" src="js/apiGateway-js-sdk/lib/apiGatewayCore/utils.js"></script>
<script type="text/javascript" src="js/apiGateway-js-sdk/apigClient.js"></script>
```

Figure 93 Implementation - Emotion Statistics (RESTful API – SDK Script Tags)

```
var dataList = []
var eventID = localStorage.getItem("eventID");

var apigClient = apigClientFactory.newClient();

apigClient.emotionsGet({}, null, {})
.then(function(result){


    dataList = result.data
    for(i=0; i<dataList.length ; i++){

        if(dataList[i]['emotion'][0]['eventID'] == eventID){
          // Intialize chart labels & labels
          var dataForMale = dataList[i]['emotion'][0]['maleEmotions'];
          var dataForFemale = dataList[i]['emotion'][0]['femaleEmotions'];
          var dataForAdults = dataList[i]['emotion'][0]['adultsEmotions'];
          var dataForChildren = dataList[i]['emotion'][0]['childrensEmotions'];
          var totalVisitorEmotions = dataList[i]['emotion'][0]['totalEmotions'];
          var overAllLabels = ['SAD', 'CALM', 'ANGRY', 'HAPPY', 'CONFUSED', 'FEAR', 'DISGUSTED', 'SURPRISED'];
```

Figure 94 Implementation - Emotion Statistics (RESTful API – SDK Implementation)

4. *Deeplens Heartbeat*

- Permissions

  The lambda function (Store-Deeplens-Status) has the write and read data permission from DynamoDB tables as well as lambda execution permissions.

Figure 95 Implementation - Deeplens Heartbeat (Permissions - Store Deeplens Status Lambda)

- Code/Programming

  Store-Deeplens-Status

  The below figure shows how attributes are being retrieved (Deeplens name and occurrence time). After obtaining these variables, they are then inserted (overwriting) values in the "Deeplens" DynamoDB table.

```
1  import json
2  import boto3
3  import uuid
4  import datetime
5
6  # dynamodb = boto3.client('dynamodb')
7  deeplenTable = 'DeepLens'
8  dynamodb = boto3.client('dynamodb')
9
10 def lambda_handler(event, context):
11
12     for eachRec in event['Records']:
13         if eachRec['eventName'] == "INSERT":
14             if 'deeplensName' in eachRec['dynamodb']['NewImage']:
15                 dlName = eachRec['dynamodb']['NewImage']['deeplensName']['S']
16                 dlLastPT = eachRec['dynamodb']['NewImage']['occurranceTime']['S']
17                 dynamodb.put_item(
18                     TableName=deeplenTable,
19                     Item={
20                         'deepLensName': {'S':dlName},
21                         'lastHeartBeat': {'S':dlLastPT}
22                     })
23
24
25
```

Figure 96 Implementation - Deeplens Heartbeat (Store Deeplens Status Lambda)

I added the below code in my colleague lambda (aggregates-dynamodb-to-s3). This code reads the “Deeplens” table and fetches the data into the web socket API; this approach is to render the last heartbeat of each Deeplens device.

```
47              tableDeepLens = dynamodb.Table(deeplensTable)
48              resp = tableDeepLens.scan()
49              deepLenses = resp['Items']
50              for eachDeepLens in deepLenses:
51                  aggregatesJSON.append(eachDeepLens)
```

Figure 97 Implementation - Deeplens Heartbeat (Web Socket API)