Bluff Detection & Prevention System (BDPS)	Document ID: LIT-420	Version: 2.1
System Design Specification (SDS)		Version Date: 11/26/2021

SOFTWARE DESIGN SPECIFICATION

FOR A Bluff Detection & Prevention Software Prototype

BY: Chavey Villegas, Eddie Coda, Eric Phan, Samara Marrow

VERSION: REVISION – 2.1

November 26, 2021

REVISION CHART / Delegation Chart

Version	Primary Author(s)	Description of Version	Date Completed
Draft	Chavey Villegas Eddie Coda Eric Phan Samara Marrow	Initial draft created for distribution and review comments	10/15/2021
Draft 2.0	Chavey Villegas Eddie Coda Eric Phan Samara Marrow	Secondary draft created to encompass **Database Management Strategy *Starts on Page 9**	11/26/2021

Primary Author(s)	Responsibilities/ Tasks	Date Completed
Chavey Villegas	- Storage in database - Interval timer going off, capturing, and running algorithm, saving image to DB	10/9/2021
Eddie Coda	- Image sent to algorithm for editing, geolocation, time, Score #, then sent to database, also may alert authorities. - Create timeline at the end with everyone tasks & roles	10/12/2021
Eric Phan	- Image comparison algorithm - Presentation of UI at central monitoring - User at central monitoring changing interval time to avoid sunrise	10/15/2021
Samara Marrow	- Networking to and from - Image alert at CM for further review, 2 images presented, one is brand new one is most recent. (Score of 4) - user deciding new score#	10/15/2021

CONTENTS

1. INTRODUCTION

1.1 Purpose

This document will define the design of the bluff detection & prevention software. It contains specific informationabout the expected input, output, classes, and functions. The interaction between the classes to meet the desired requirements are outlined in detailed figures at the end of the document.

1.2 Scope

This Design Specification is to be used by Software Engineering and Software Quality Engineering as a definition of the design to be used to implement the prototype bluff detection & prevention software system.

1.3 Objective

The Bluff Detection & Prevention System (BDPS) will be a completely automated, Surveillance-based detection system monitoring bluffs, and alerting authorities near Del Mar in effort to prevent potential harm. It will take advantage of the Internet and Wi-Fi-enabled cameras to monitor staged areas and send alerts out if necessary. Utilizing Image comparison algorithms and supporting databases, the system will be able to record a change, decide a rating for an image, store and recall images, as well as send alerts.

2. System Overview

2.1 Product Perspective

This product is a prototype to evaluate the feasibility of this system for further use on the entire California coastline. The BDPS will capture 50' x 50' images of the top of bluffs resulting in 4,000 pixels per SqFt. The Images will be sent back to a central monitoring site via Wi-Fi networks and cloud-based software. It will include a time stamp and geolocation tagged on each image. Once the image is sent back to central monitoring, a specialized image comparison software will decide a rating for the cliff change. Depending on the values that are assigned to each rating, the software will decide to either store the image, or store the image and send out alerts to authorities, including Lifeguards and Amtrack rail operators. Ratings listed as follows:

Enable image comparison algorithms to determine a rating value listed below (0 – 5).

(Rating value: 0)
- *Indicates no change, no need for alert.*

(Rating value: 1 - 3)
- *Considered predictive of a significant slide.*

(Rating value: 4)
- *Significant change requires immediate evaluation to determine safety protocols.*
- *Alerts sent out immediately to both Lifeguards and rail operators.*

(Rating value: 5)
- *Major change has occurred.*
- *Alerts sent out immediately to both Lifeguards and rail operators.*
- *Image indicating major change should be compared to the last image to determine if tragedy to beachgoers or railway occurred during the incident.*

2.1.1 Design Method

The design of this product utilizes an object-oriented approach.

2.1.1 User Interfaces

*The user of this software product will be interfacing with the image comparison algorithm when images need further review. The users of Lifeguards and Amtrack operators will only be interfacing with an update alert via **RedFlag**, which is the server system that sends real-time mass notifications to people when and where they need it.*

2.1.2 Hardware Interfaces

This prototype software can operate on most Arlo wireless cameras.

2.1.3 Software Interfaces

This system will execute on an AWS server for better use in database, scalability, pricing, networking, and cloud-based serves. It will have an Ubuntu LINUX platform running [Eclipse or VS] compiler with OpenCV.

.

2.1.4 Memory Constraints

This software and algorithmic backed database take up about 7TB of memory for one months' worth of images. The output reports are modest in size and take upabout 7kb.

Our Data Management Strategy listed below will cover the importance of our data strategy, including these key factors:

- *Slow and inefficient business processes*
- *Data privacy, data integrity, and data quality issues that undercut your ability to analyze data*
- *Lack of deep understanding of critical parts of the business (customers, supply chain, competitive landscape, etc.) and the processes that make them tick*
- *A lack of clarity about current business needs (a problem that descriptive analytics can help solve) and goals (which predictive and prescriptive analytics can help identify)*
- *Inefficient movement of data between different parts of the business, or duplication of data by multiple business units*

2.1.5 Operations

The operator will be required to enter the initial time interval for the cameras in the system and respond to further evaluation of images if necessary (rating of 4).

2.1.6 System Adaptation Requirements

- *This software is intended to execute on an Ubuntu LINUX platform with no modifications needed to support different sites.*
- *Deciding between using Python vs C++ vs JS due to the support for OpenCV image processing algorithm: Winner is JS*
  - *If what you write is intended for users other than yourself, you will need a proper GUI. That usually spells out a JS library (e.g. wxWidgets for cross platform).*
  - *Performance with JS is considerably better than Python when dealing with simplicity and documentation with AWS. If, for example, your application requires real-time image processing, JS is pretty much the only option.*
  - *What if the IP/CV stuff is being coded into embedded devices? What if the processing and response time are very critical and the application is real time? We'd prefer using JS integration (in fact most people use JS now as it offers objective orientation yet being good enough fast).*

2.2 Product Functions

Confidence Level -This function calculates the % change in images and gives root to the main algorithm in use, utilizing Histograms and greyscale to analyze changes in pixels within the image.

Handler-This function Handler handles create, read, update, and remove requests for all microservices to implement.
Alert API-This internal API confirms send and receive messages with the RedHat external software that we chose to outsource the action of sending alerts.

2.3 User Characteristics

The general characteristics of the intended users, include

- *educational level –Bachelor of Science*
- *experience- Division of geological and marine biology survey*

2.4 Constraints

This application can only run on a system that supports Open-CV Eclipse compiler.

2.5 Assumptions and Dependencies

This system is a prototype, using full capabilities, on a smaller scale.

2.6 Apportioning of Requirements

There are requirements yet determined apportioned to later releases of the system.

3. DESIGN CONSIDERATIONS

3.1 Operating Environment

The BDPS is intended to be operated ina AWS service hosting an Ubuntu Linux environment with an Eclipse Compiler.

3.2 Fault Tolerant Design

Application errors will be handled by common fault detection services(e.g. common exception handling, and error checking on task processing).

3.3 Design Conventions

The BDPS software design uses the Object-Oriented methodology described in "The Unified Software Development Process" by Ivar Jacobsen, Grady Booch and James Rumbaugh. (Booch, 1999)

3.4 Architectural Design

The software capabilities and requirements specified in BDPS Software Requirements Specification are transformed into programs that will execute on an Ubuntu system running (Eclipse or VS). Software items are partitioned into classes and packages using Object Oriented methodology to maximize encapsulation and minimize interfaces. Packages are then built (compiled and linked) into executable programs

3.5 Database Management Strategy

3.5.1 Overview: Using DynamoDB

The DynamoDB database provides an easy to configure, high-performance, SQL database with low operational overhead and extreme scalability. It appeals to developers with OLTP applications requiring a simple serverless database or those requiring the utmost in scalability.

DynamoDB gives the option to use both NoSQL and SQL (relational) data management, due to the necessity for run-time queries and large-scale consistency due to the volume of images that will be stores and called upon frequently.

More recently, best practices have evolved around DynamoDB single-table design patterns where one database table serves the entire application and holds multiple different application entities. This design pattern offers greater performance by reducing the number of requests required to retrieve information and lowers operational overhead. It also greatly simplifies the changing and evolving of your DynamoDB designs by uncoupling the entity key fields and attributes from the physical table structure.

3.5.2 Data Management Strategy for BDPS: (4 Components)

3.5.2.1 (1) Business Strategy:

Achieve continuous data quality, meaning that the company will identify and address data quality problems continually, rather than relying on periodic checks.

Design or Implementation of DynamoDB should be normalized then optimized to handle the volume of images and meet accepted performance requirements.
Regarding the price of DynanamoDB we have constructed a price table based on the average reads we predict to see:
- Assume your table occupies 25 GB of storage at the beginning of the month and grows to 29 GB by the end of the month, averaging 27 GB based on continuous monitoring of your table size. The first 25 GB of storage are included in the AWS Free Tier. The remaining 2 GB of storage are charged at $0.25 per GB, resulting in a table storage cost of $0.50 for the month:

Timeframe (Day of Month)	Total Writes	Total Reads
1–7	7,000 writes (1,000 writes x 7 days)	7,000 reads (1,000 reads x 7 days)
8	10,000,000 writes	10,000,000 reads
9–22	2,100,000 writes (150,000 writes x 14 days)	2,100,000 reads (150,000 reads x 14 days)
23–29	70,000 writes (10,000 writes x 7 days)	70,000 reads (10,000 reads x 7 days)
30	30,000,000 writes	30,000,000 reads
Monthly total	42,177,000 writes	42,177,000 reads
Monthly bill	$52.72 ($1.25 per million writes x 42.177 million writes)	$10.54 ($0.25 per million reads x 42 million reads)

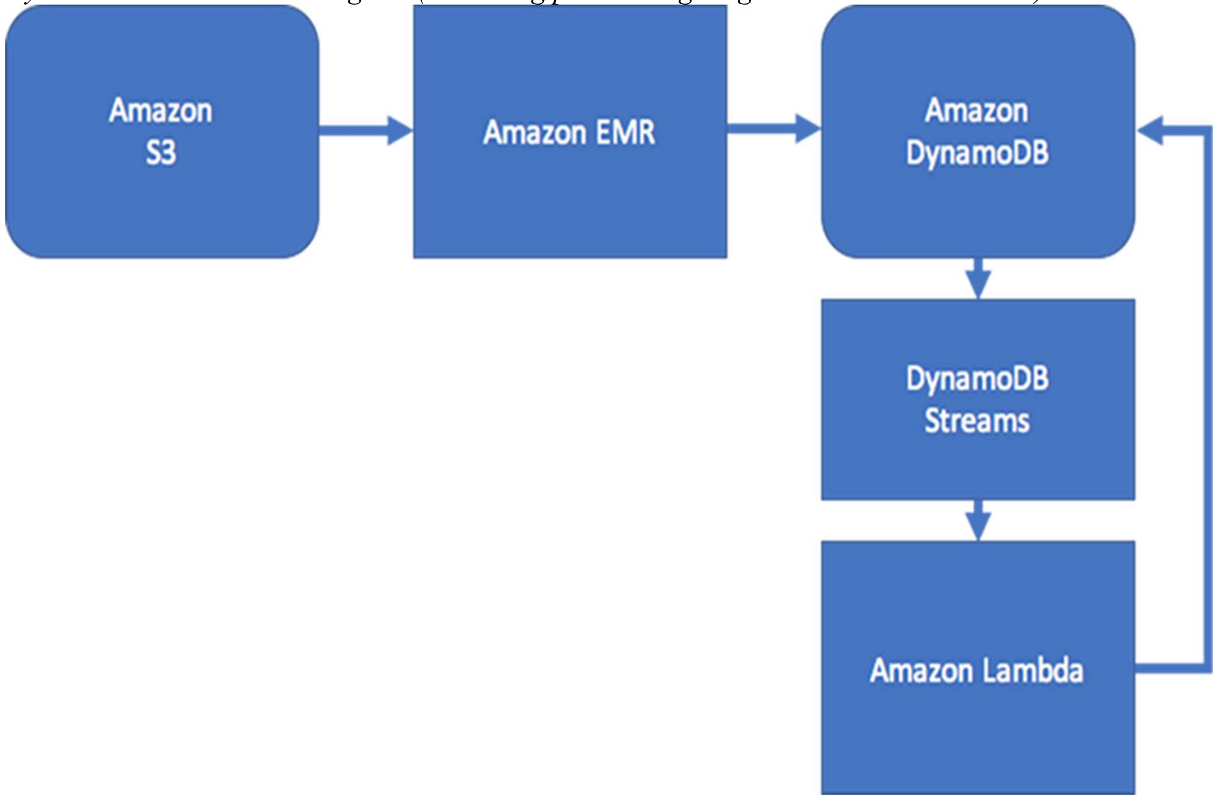
3.5.2.2 (2) Organizational Roles:

Three main types of users typically implement and enforce data strategy:

- *Data engineers, who oversee the data pipeline and are responsible for building an efficient, reliable data architecture.*
- *Data scientists, who work with data that the pipeline delivers including images and geolocation tags.*
- *Data analysts, who specialize in analyzing and interpreting data to better understand the numbering process of the BDPS.*
- *Business managers, who help to manage data operations and review data reports.*

3.5.2.3 (3) Data Architecture:

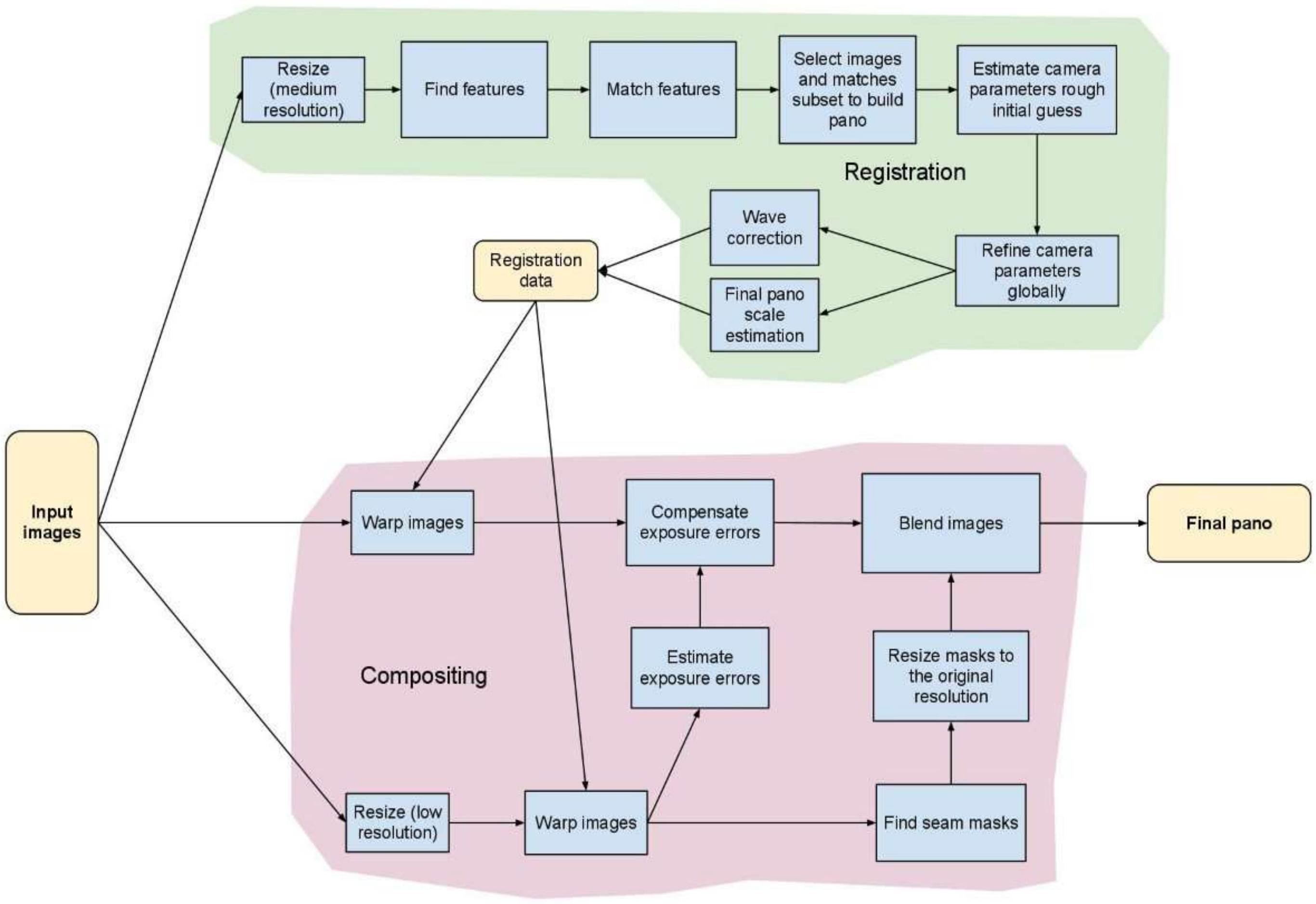
DynamoDB Architecture Diagram (Extending pre-existing diagram listed in section 4.1)



Basic Logic Table of Database Operations regarding Testing:

Characteristic	Amazon DynamoDB
Tools for Accessing the Database	You can use the Ubuntu Console or the Command Line Interface to send ad hoc requests to DynamoDB and view the results.
Connecting to the Database	DynamoDB is a web service, and interactions with it are stateless. Applications do not need to maintain persistent network connections. Instead, interaction with DynamoDB occurs using HTTP(S) requests and responses.
Authentication	Every request to DynamoDB must be accompanied by a cryptographic signature, which authenticates that particular request.
Authorization	The BDPS authorization is handled with the integration of our internal API “Alert”. You can write a policy to grant permissions on a DynamoDB resource (such as a table), and then allow users and roles to use that policy.
Sending a Request	The Internal API handler sends HTTP(S) requests to DynamoDB. The requests contain the name of the DynamoDB operation to perform, along with parameters. DynamoDB runs the request immediately.
Receiving a Response	DynamoDB returns an HTTP(S) response containing the results of the operation. If there is an error, DynamoDB returns an HTTP error status and messages.

DynamoDB Stitching-Pipeline design for the BDPS:


Resize (medium resolution)
Find features
Match features
Select images and matches subset to build pano
Estimate camera parameters rough initial guess
Registration
Wave correction
Registration data
Refine camera parameters globally
Final pano scale estimation
Input images
Warp images
Compensate exposure errors
Blend images
Final pano
Estimate exposure errors
Resize masks to the original resolution
Compositing
Resize (low resolution)
Warp images
Find seam masks

3.5.2.4 (4) Data Management

3.5.2.4.1 Here we will describe the relational data in DynamoDB to Maximize performance. Entity relationship explanations below:

In SQL, designing models are based on database normalization. We design our data according to these laws, first normal form, second normal form, and third normal form.

However, understanding access pattern is key when designing a data model in DynamoDB. For instance, we design our partition key and sort key based on how the user usually seeks operation. There is no joining in DynamoDB because joining cannot tolerate their performant use case; yet a common way to model data in DynamoDB is to apply relational design patterns. They put their items in a different table and do joins on the application level. This "joins" operation becomes the bottleneck as Network I/O is the slowest and cannot perform in a concurrent environment.

One tip of designing your model in your DynamoDB is to have the application handle as few requests to DynamoDB as possible - ideally one.

Based on the design example we have built, these are the access patterns we will need to implement:

- *User should be able to search for images based on geolocation*
- *User should be able to search for images based on timestamp*
- *User should be able to search for images based on photoKey*
- *User should be able to search for images based on ratingReview*

If we want to design this in a relational database, we will have four tables: geolocation, timestamp, rating #, and a join table for photoKey tags. Images contain a foreign key that associates with photoKey. The join table contains geolocation associate with timestamp. There is a one-to-many relationship between geolocation and timestamp (a location can have many stamped images on different times). There is a one-to-many relationship between photokey and ratingReview (a ratingReview can have multiple images with photoKeys)

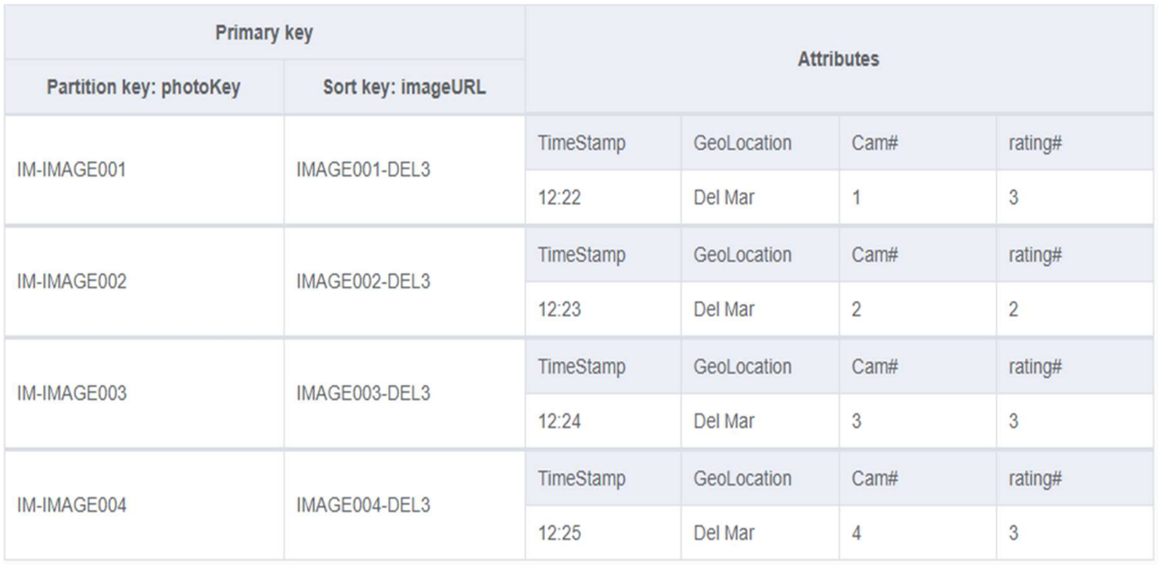
Before we model a diagram we need to explicitly describe the access patterns:

Access Pattern
Get all the photoKeys for a given geolocation
Get all the timestamps for a given geolocation
Get all the photoKeys for a given ratingReview(imageURL)
Get all the time stamps for a given ratingReview(imageURL)
Get all the geolocations for a given ratingReview(imageURL)
Update a ratingReview(imageURL)

Now that we have the access patterns, we can build the model as such:

#	Entity	Description	Parameters
1	User	Update	
2	User	Get all	
	User	Delete	
3	Image	Get by photoKey	String
4	Image	Get by timestamp	String
5	Image	Get by geolocation	String

Because an item in DynamoDB must be uniquely identified by its primary key, the sort key will be the way we differentiate an image from its tags. Let's have a look at the following primary key structure:



Primary key		Attributes			
Partition key: photoKey	Sort key: imageURL				
IM-IMAGE001	IMAGE001-DEL3	TimeStamp	GeoLocation	Cam#	rating#
		12:22	Del Mar	1	3
IM-IMAGE002	IMAGE002-DEL3	TimeStamp	GeoLocation	Cam#	rating#
		12:23	Del Mar	2	2
IM-IMAGE003	IMAGE003-DEL3	TimeStamp	GeoLocation	Cam#	rating#
		12:24	Del Mar	3	3
IM-IMAGE004	IMAGE004-DEL3	TimeStamp	GeoLocation	Cam#	rating#
		12:25	Del Mar	4	3

This table represents one portion of data management to be implemented as such. The remaining tables will be build at a future time once more requirements and specifications are laid out by developers.

3.6 User Interface:

3.6.1 Overview:

The users will be taken to the Overview page after they are authenticated with access permissions to the system. Here, they can view the network status of the cameras and modify the intervals, image specifications, and firmware if there are any updates available. If a camera is detected as "Offline", the user will have the option to attempt reestablishing connection or contact the site administrator for in-person troubleshooting.

3.6.2 Image Review:

When the image comparison algorithm detects a bluff change, images will be added to the "Further Review" list with a rating of four, where the user can further evaluate if traffic should be stopped. After reviewing, the user can submit the appropriate rating for the bluff. If a rating of 5 is submitted by the user, the image will be compared to the last image captured at the location to determine if rescue operations need to be held

3.6.3 Image Database:

The system provides a page where users can view the entire image database hosted using DynamoDB. Each image will provide information on the geolocation, timestamp, and camera that captured the image. For organization, users are given the ability to sort by their preference, with the newest set as the default setting.

3.6.4 Network:

All surveillance cameras will capture any bluff detections, a wireless data transfer will take place to transfer the footage into the AWS cloud through the Amazon API Gateway, and the AWS Lambda function, which will then trigger and notify SNS. The Lambda function interacts with DynamoDB and returns a response to API Gateway.

The client will be able to stream the video surveillance of any bluff detections on their computers through the internet.

3.6.5 UI EXAMPLES:

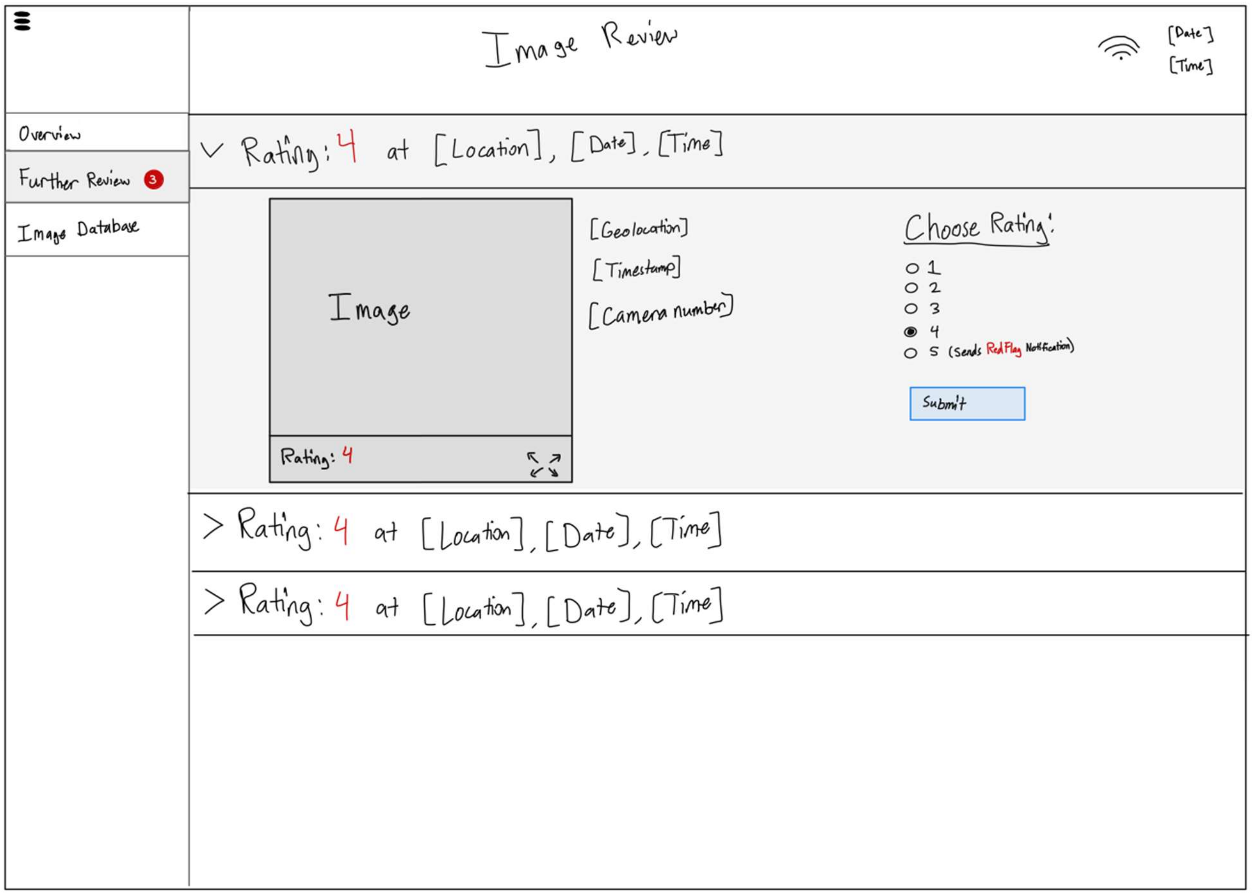
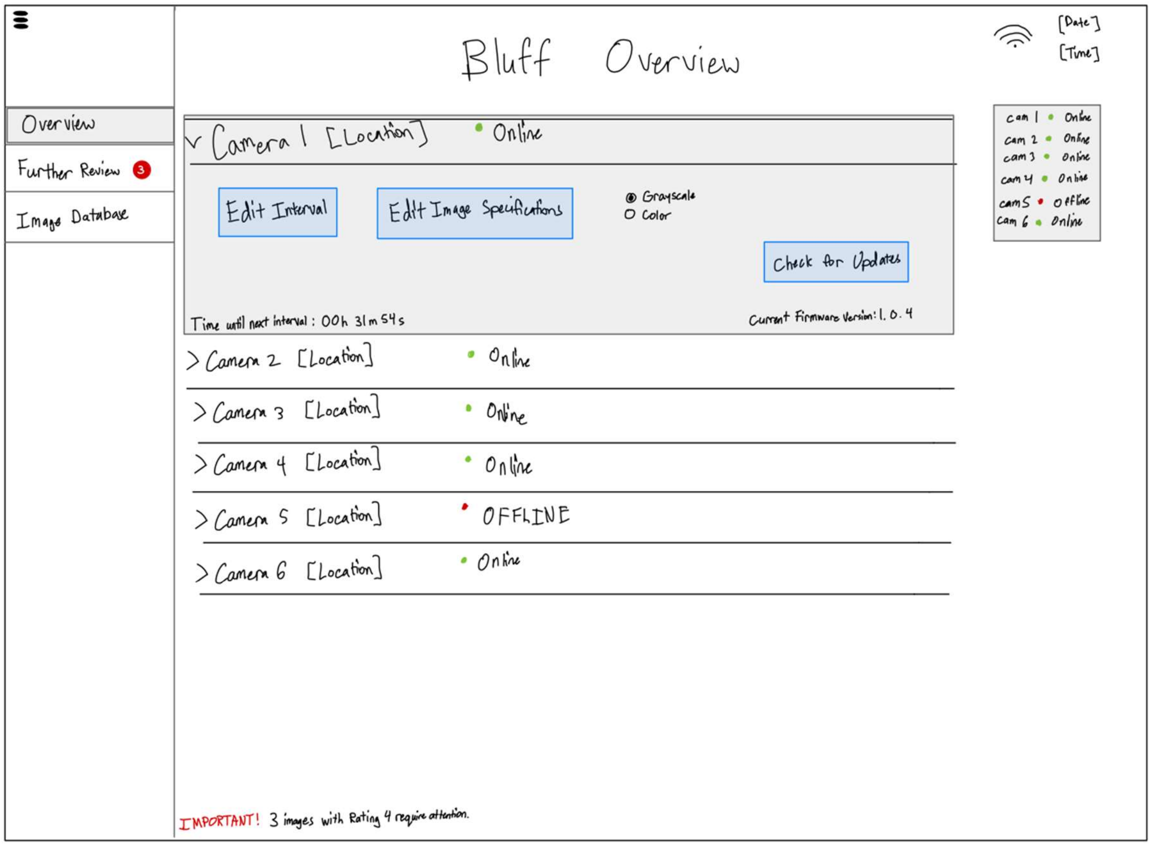


Image Review
[Date]
[Time]
Overview
Further Review 3
Image Database
Rating: 4 at [Location], [Date], [Time]
Image
Rating: 4
[Geolocation]
[Timestamp]
[Camera number]
Choose Rating:
1
2
3
4
5 (Sends Red Flag Notification)
Submit
Rating: 4 at [Location], [Date], [Time]
Rating: 4 at [Location], [Date], [Time]


Bluff Overview
[Date]
[Time]
Overview
Further Review 3
Image Database
Camera 1 [Location] • Online
Edit Interval
Edit Image Specifications
Grayscale
Color
Check for Updates
Time until next interval: 00h 31m 54s
Current Firmware Version: 1.0.4
Camera 2 [Location] • Online
Camera 3 [Location] • Online
Camera 4 [Location] • Online
Camera 5 [Location] • OFFLINE
Camera 6 [Location] • Online
cam 1 • Online
cam 2 • Online
cam 3 • Online
cam 4 • Online
cam 5 • offline
cam 6 • Online
IMPORTANT! 3 images with Rating 4 require attention.

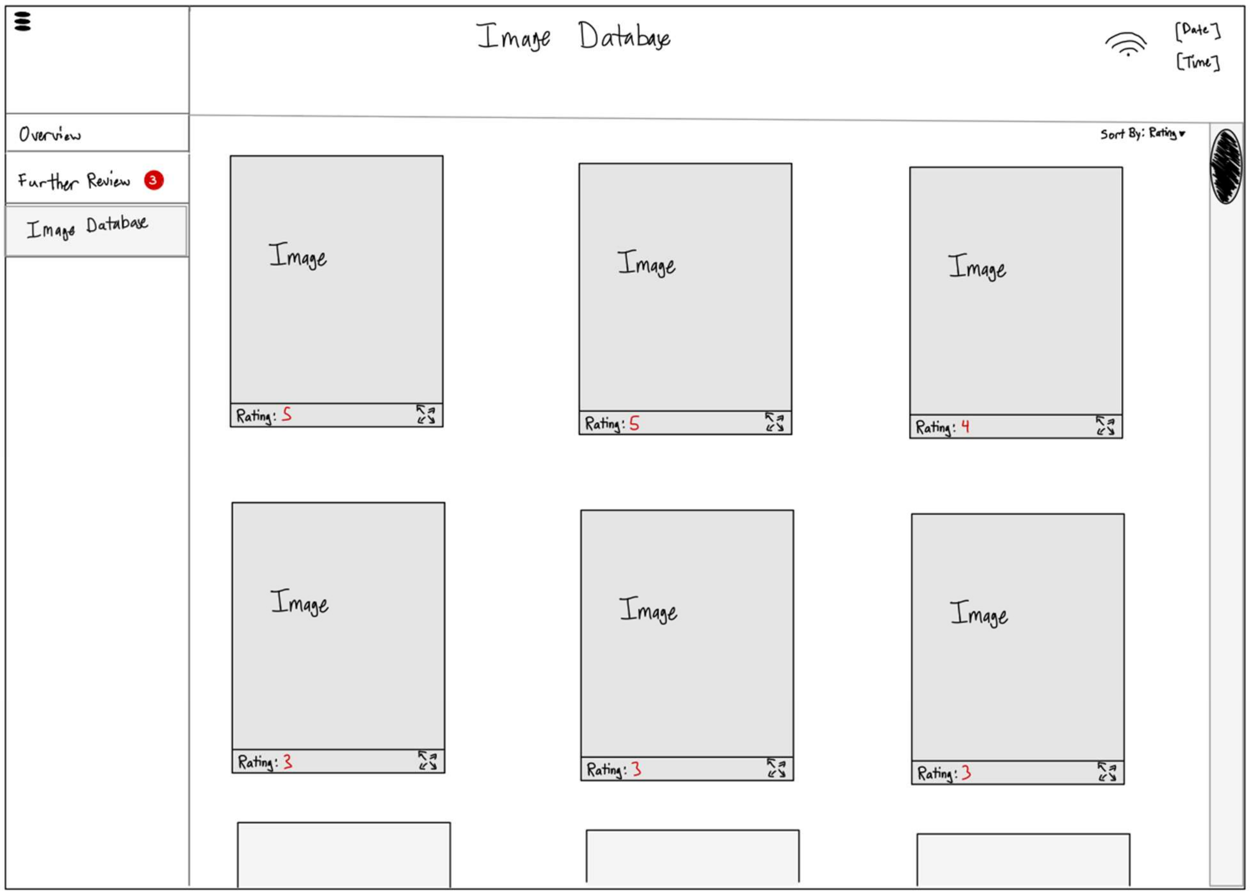
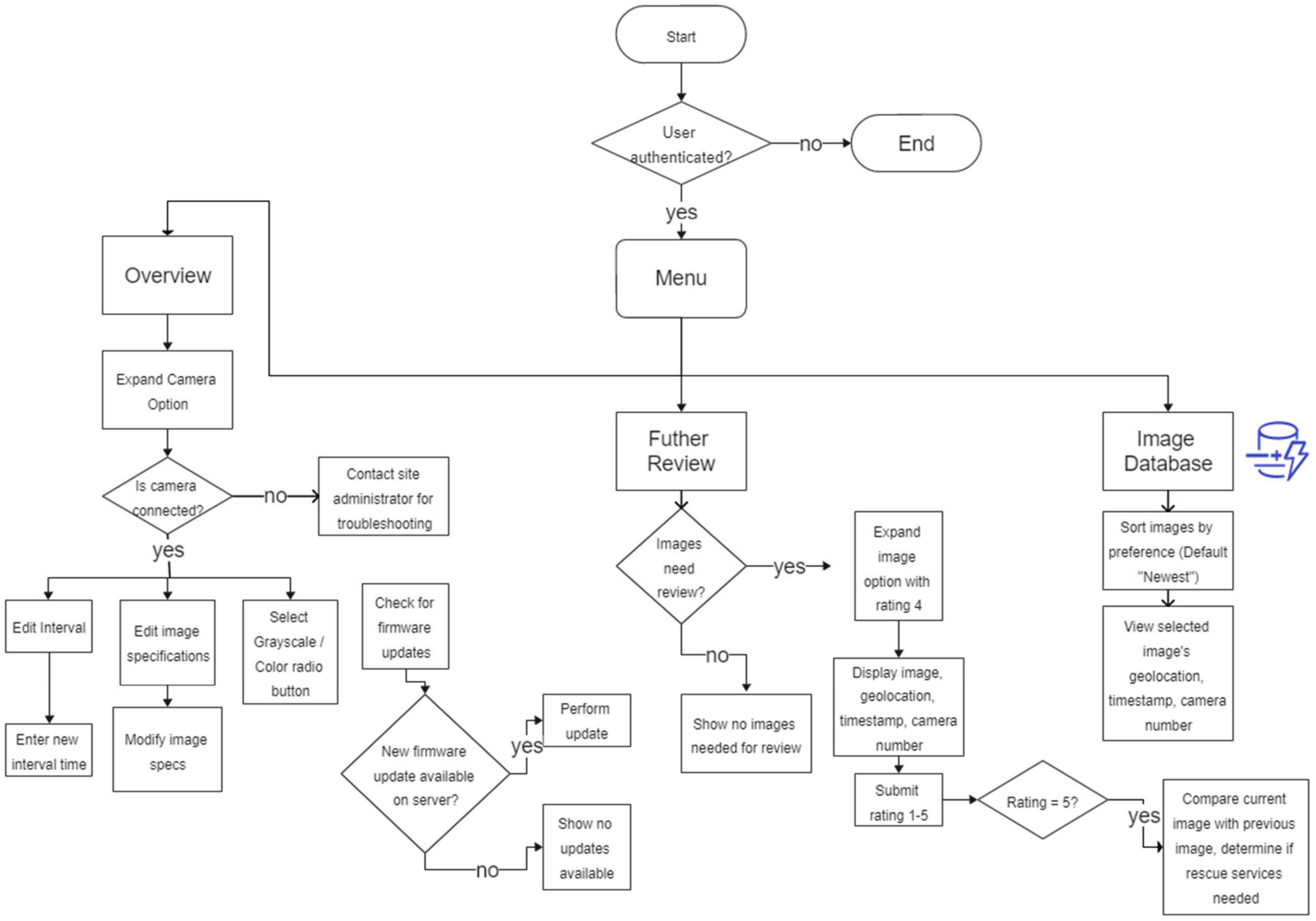
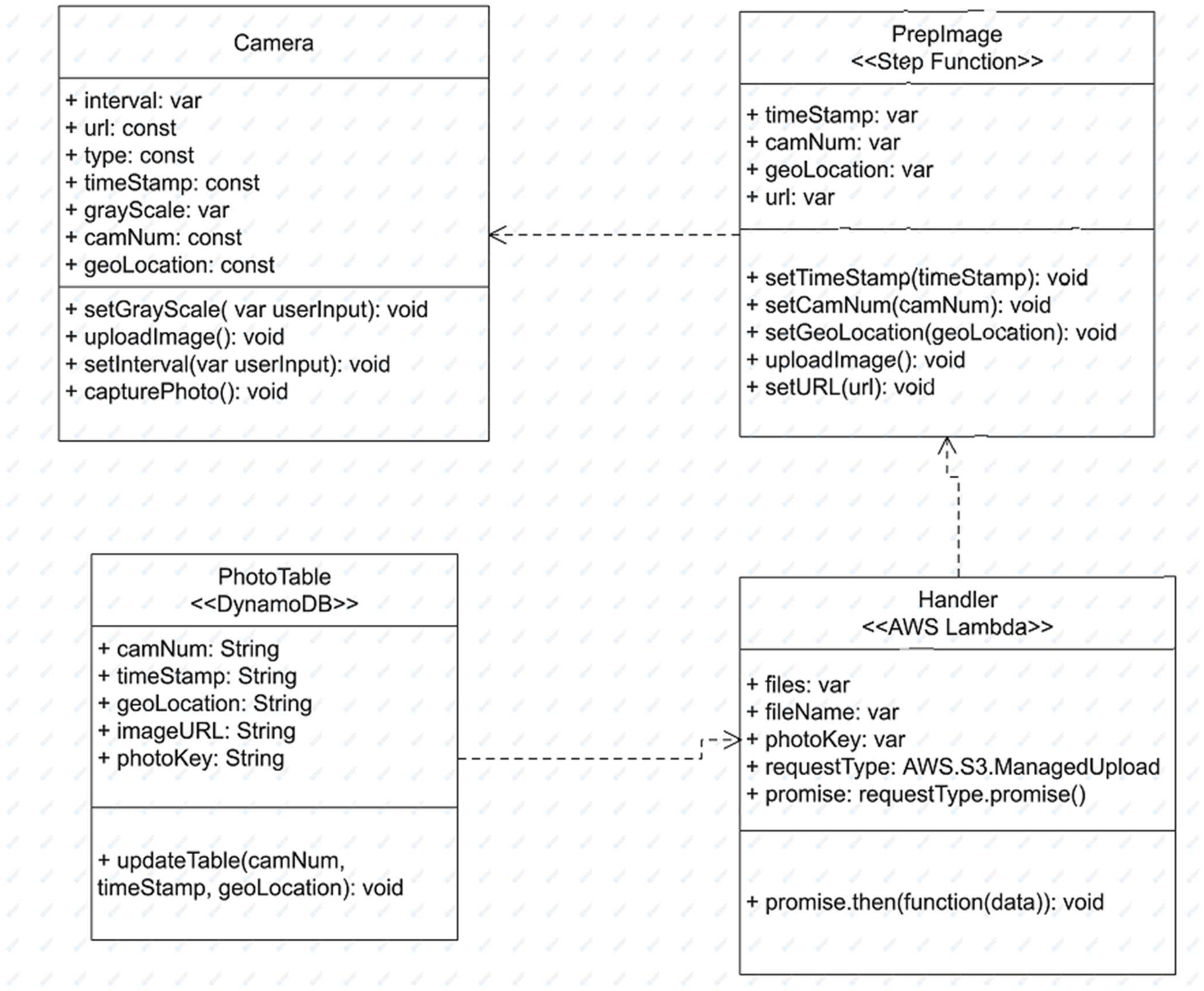


Image Database
[Date]
[Time]
Overview
Further Review
3
Image Database
Sort By: Rating
Image
Rating: 5
Image
Rating: 5
Image
Rating: 4
Image
Rating: 3
Image
Rating: 3
Image
Rating: 3


Start
User authenticated?
no
End
yes
Menu
Overview
Expand Camera Option
Is camera connected?
no
Contact site administrator for troubleshooting
yes
Edit Interval
Enter new interval time
Edit image specifications
Modify image specs
Select Grayscale / Color radio button
Check for firmware updates
New firmware update available on server?
yes
Perform update
no
Show no updates available
Futher Review
Images need review?
yes
Expand image option with rating 4
Display image, geolocation, timestamp, camera number
Submit rating 1-5
Rating = 5?
yes
Compare current image with previous image, determine if rescue services needed
no
Show no images needed for review
Image Database
Sort images by preference (Default "Newest")
View selected image's geolocation, timestamp, camera number

4. System Architecture

4.1 View of Product Classes (Figure 1)



<u>Figure 1- UML Main Class structures examples</u>

This figure shows the overall design of the major components in our system with outlining the individual classes and theirrelation to each other.

4.2 Individual Classes of System

4.2.1 Camera:

The camera class is where the camera is being controlled. The class stores the required information associated with pictures of different sections of the bluff. Camera utilized seven attributes. Interval and Grayscale are of type var. URL, type, camNum, and geoLocation are of type const. Camera also utilizes four functions setGrayScale, uploadImage, setInterval, and capturePhoto.

4.2.1.1 Attributes of Camera class

4.2.1.1.1 INTERVAL
Holds the time between photos taken by the camera. The default time is one hour.

4.2.1.1.2 URL
Holds the location of the image taken by the camera.

4.2.1.1.3 timeStamp
Holds the date and time of the picture that was taken by the camera.

4.2.1.1.4 greyScale
Holds true if the user wants photo to be in grayscale.

4.2.1.1.5 camNum
Holds the camera number.

4.2.1.1.6 geoLocation
Holds the geographical information of the camera that took the picture.

4.2.1.2 Functions available in Camera Class

4.2.1.2.1 setGreyscale(var userInput)
This function takes a user input to set the attribute greyScale to true or false.
The user input type is var.

4.2.1.2.2 setInterval(var userInput)
This function sets the length of time between each photo taken by the camera. It takes a user input of type var.

4.2.1.2.3 capturePhotoData()
This function instructs the camera to take a photo

4.2.1.2.4 uploadImage( )
This function makes a POST request to the PrepImage class. The information contained in the header sent is url, type, timeStamp, camNum, and geoLocation.

4.2.2 PrepImage

This class prepares the image to be added to the database. PrepImage has four attributes. timeStamp, camNum, geoLocation, and url. All four are of type var. This class also utilizes four functions. setTimeStamp, setCamNum, setGeoLocation, and uploadImage..

4.2.2.1 Atributes of PrepImage Class

4.2.2.1.1 timeStamp:

Holds the date and time of the photo that is being prepared for the database.

4.2.2.1.2 camNum:

Holds the camera number of the photo that is being prepared for the database.

4.2.2.1.3 geoLocation:

Holds the geographical information of the photo that is being prepared for the database.

4.2.2.1.4 url:

Holds the image url of the photo that will be sent to the database.

4.2.2.2 Functions available in PrepImage Class

4.2.2.2.1 setTImeStamp(timeStamp)

This function sets the date and time for the attribute timeStamp. This function takes the timeStamp sent from the Camera class as its input.

4.2.2.2.2 setCamNum(camNum)

This sets the camNum attribute to the camNum that was passed in by the Camera class . This function takes the camNum sent from the Camera class as its input.

4.2.2.2.3 setGeoLocation(geoLocation(geoLocation)

This sets the geoLocation attribute to the geoLocation that was passed in by the Camera class. This function takes the geoLocation sent from the Camera class as its input.

4.2.2.2.4 setURL(url)

This function sets the url attribute to the url that was passed in by the Camera class. This function takes the url sent from the Camera class as its input

4.2.2.2.5 uploadImage( )

This function makes a POST request to the RESTFUL API for the Handler class. The attributes timeStamp, camNum, geoLocation, and url are sent in the header.

4.2.3 Handler

This class handles the different requests made to the server. Handler handles create, read, update, and remove requests. This class has five attributes. files, fileName, photoKey, requestType, and promise. files and filename are of type var. requestType is an AWS.S3 object. promise is a promise object. The Handler class utilizes one function promise.then().

4.2.3.1 Attributes of Handler Class

4.2.3.1.1 Files

This attribute holds the image being processed

4.2.3.1.2 fileName

Holds the name of the photo being processed

4.2.3.1.3 photoKey

Holds the key associated with the photo being processed

4.2.3.1.4 requestType

This object determines what type of request to make to the database. The possible choices are create, read, upload, and delete

4.2.3.1.5 promise

This attribute contains the promise object required for asynchronous requests

4.2.3.2 Functions available in Handler Class

4.2.3.2.1 promise.then(function(data))

This class initiates the promise function to make a request to the database. It takes the data of the handler class as an input.

4.2.4 PhotoTable

This is the class inside of the database. It handles create, read, update, and delete requests. The class creates a table within the database to store all information. PhotoTable has five attributes. camNum, timeStamp, geoLocation, imageURL, and photoKey. All five attributes are of type String. PhotoTable utilizes four functions. createTable, readTable, updateTable, and deleteTable..

4.2.4.1 Attributes for PhotoClass

4.2.4.1.1 camNum()

This attribute holds the camNum that was provided by the Handler class

4.2.4.1.2 timeStamp()

This attribute holds the timeStamp that was provided by the Handler class.

4.2.4.1.3 geoLocation()

This attribute holds the geoLocation that was provided by the Handler class.

4.2.4.1.4 photoKey()

This attribute holds the photoKey that was provided by the Handler class

4.2.4.1.5 ImageURL()

This attribute holds the url that was provided by the Handler class.

4.2.4.2 Functions available in PhotoTable class

4.2.4.2.1 createTable(camNum, timeStamp, geoLocation, imageURL, photoKey( )

This function creates a new table within the database. It takes the camNum, timeStamp, geoLocation, imageURL, and photoKey attributes and sets the values within the new table.

4.2.4.2.2 readTable ( )

This function returns the contents of an existing table. If there is no table present, the function returns NULL.

4.2.4.2.3 updateTable(camNum, timeStamp, geoLocation, imageURL, photoKey)

This function edits existing items in a table. photoKey is a required argument. The user can update the camera number, timestamp, geolocation, and imageURL. These arguments are optional

4.2.4.2.4 deleteTable(photoKey( )

This function deletes an item in the table. The photoKey passed as an argument will be removed from the table.

5. Figures

5.1 Use Cases (Figure 2)

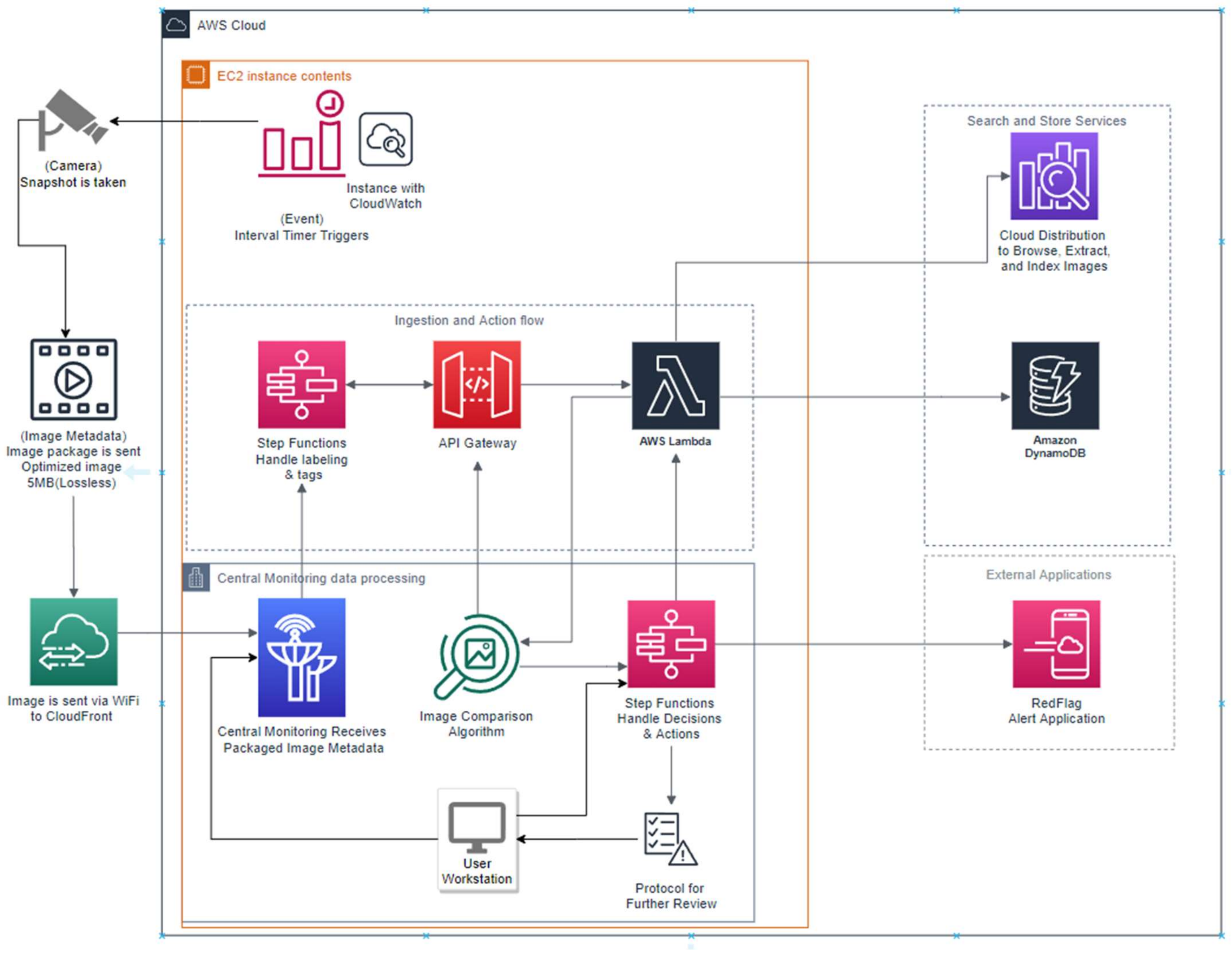


Figure 2- Interval Timer triggers Image Capture Protocol

5.2 Image Comparison Algorithm

5.2.1 Multi-level Architectural Design (Figure 3)

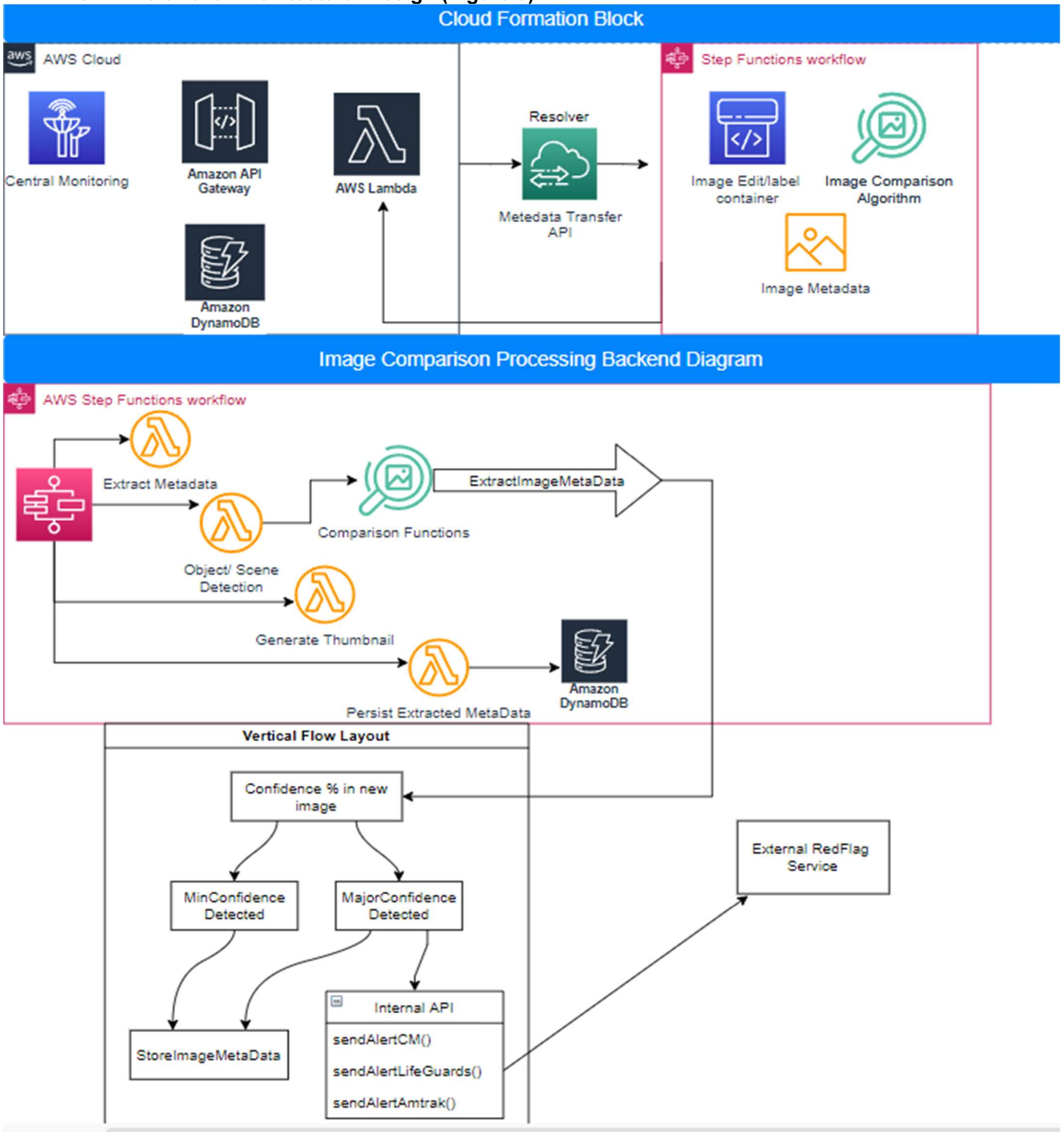


Figure 3- Image Comparison Algorithm

5.2.2 Networking Flow from Cameras (Figure 4)

5.2.2.1 Flow from Camera Station to CM

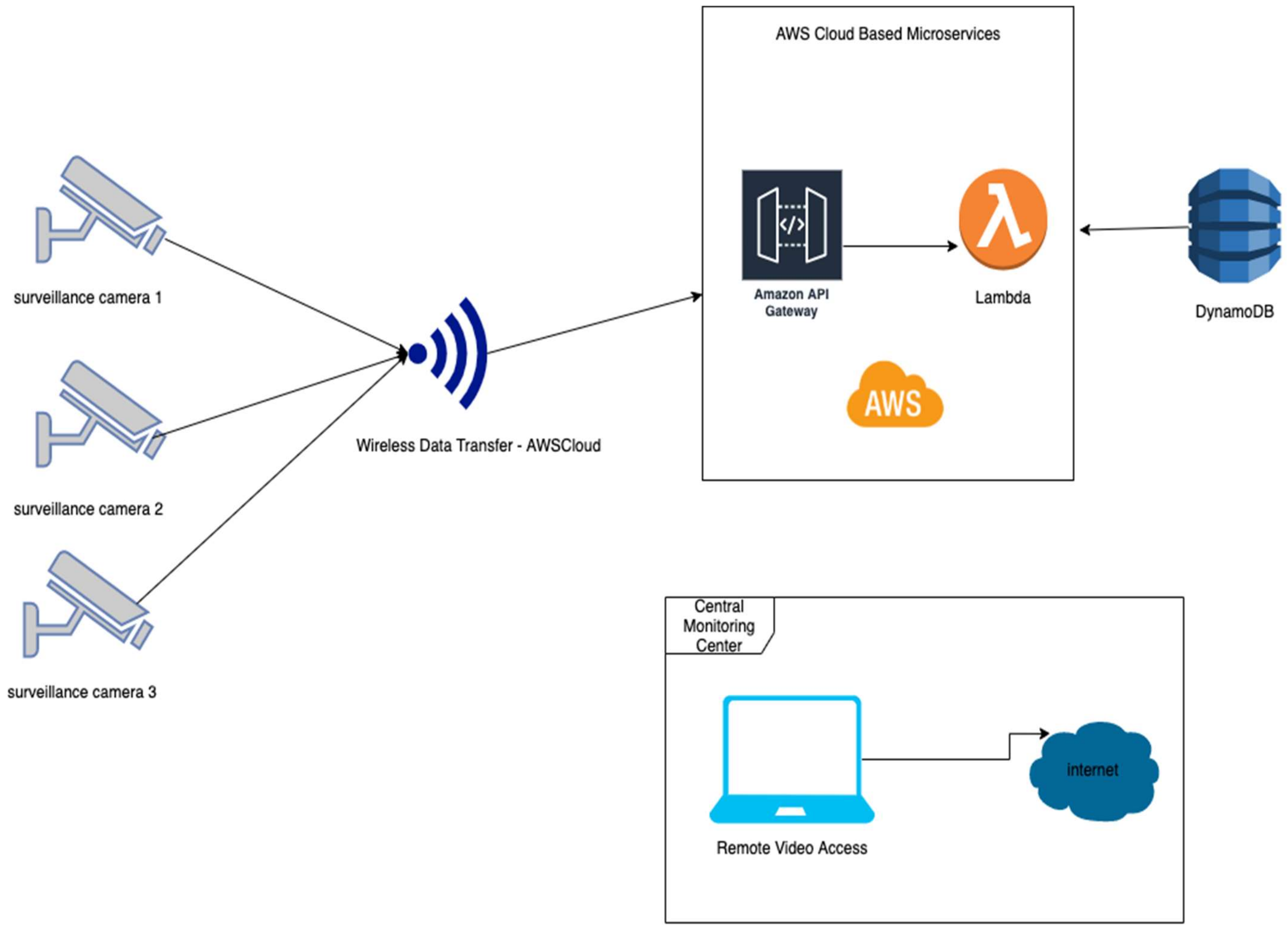


Figure 4- Camera Networking Flow

6. REFERENCES

6.1 References

Appleton, Brad . A Software Design Specification Template. N.d. <http://www.enteract.com/~bradapp/docs/sdd.html>.

Docs.aws.amazon.com. 2021. Overview of AWS SDK Support for DynamoDB - Amazon DynamoDB. [online] Available at: <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Programming.SDKOverview.html> [Accessed 17 October 2021].

Amazon Web Services. 2021. Fast and Cost-Effective Image Manipulation with Serverless Image Handler | Amazon Web Services. [online] Available at: <https://aws.amazon.com/blogs/architecture/fast-and-cost-effective-image-manipulation-with-serverless-image-handler/> [Accessed 17 October 2021].

Docs.aws.amazon.com. 2021. Exporting and Importing DynamoDB Data Using AWS Data Pipeline - Amazon DynamoDB. [online] Available at: <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DynamoDBPipeline.html> [Accessed 17 October 2021].

Booch, Grady, Ivar Jacobsen, and James Rumbaugh. The Unified Software Development Process (The Addison-Wesley Object Technology Series). 1st. Ed. New York: Addison Wesley, 1999..

GCC Home Page - GNU Project . Free Software Foundation. N.d. <http://gcc.gnu.org/>.

Sommerville, Ian. Software Engineering. 6th. Ed. New York: Addison Wesley, 2001.