“Unstable Bluff” Detection System Design 2.0 with Data Management Strategy

Prepared by: “The Bluffers” - Samantha James, Joshua Olaoye, Antonio Rivera

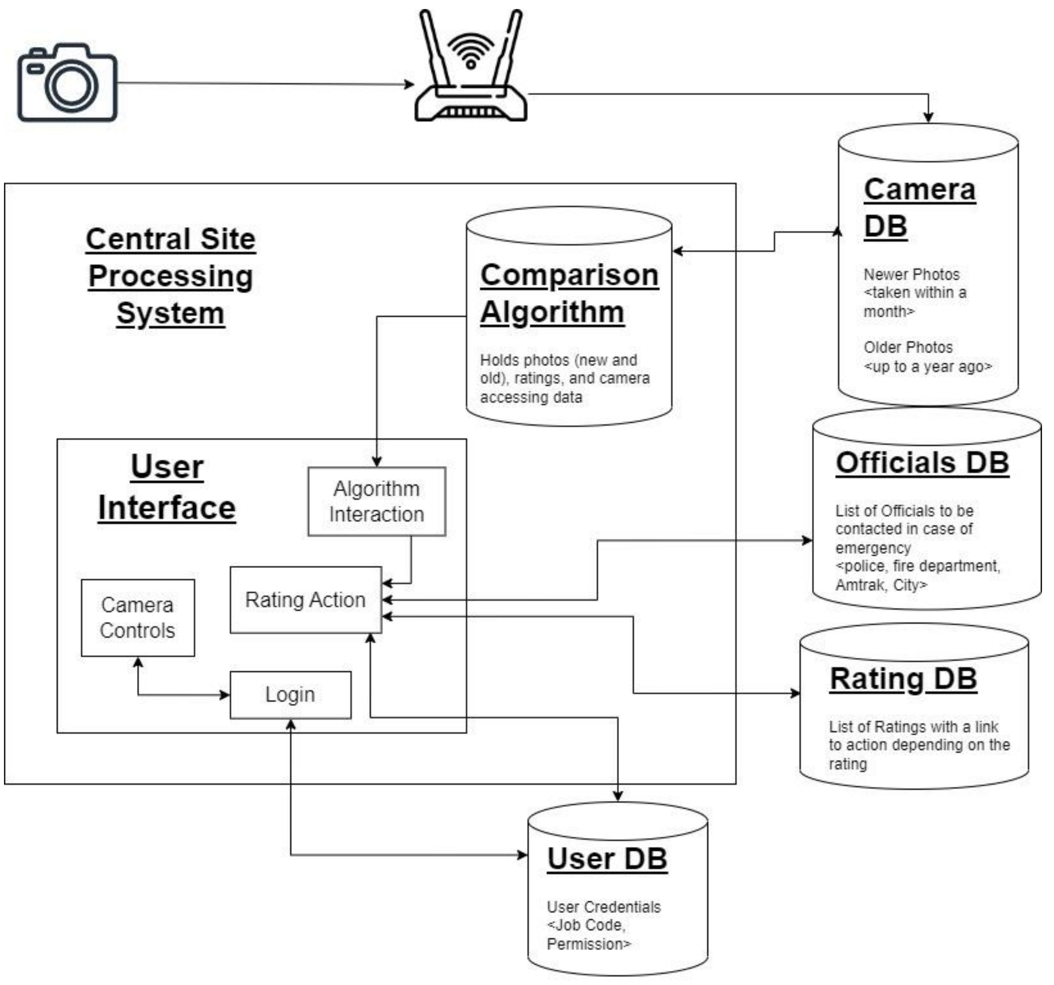
April 21, 2023

System Description

There are unstable bluffs, which are steep cliffs or banks, in the city of Del Mar along California's southern coast. A heavily trafficked train's tracks are located on a portion of these bluffs, which is very dangerous if these bluffs' stability is not monitored, potentially leading to injury and/or death of those on the train as well as people on the beach below the bluffs. This camera-based monitoring system's purpose is to constantly monitor the bluffs and detect when areas become unstable and alert authorities so that safety precautions can be implemented. This document will outline the software and architecture necessary for designing and later implementing this bluff monitoring system, which is divided into sections of software architecture overview and development plan and timeline.

Software Architecture Overview

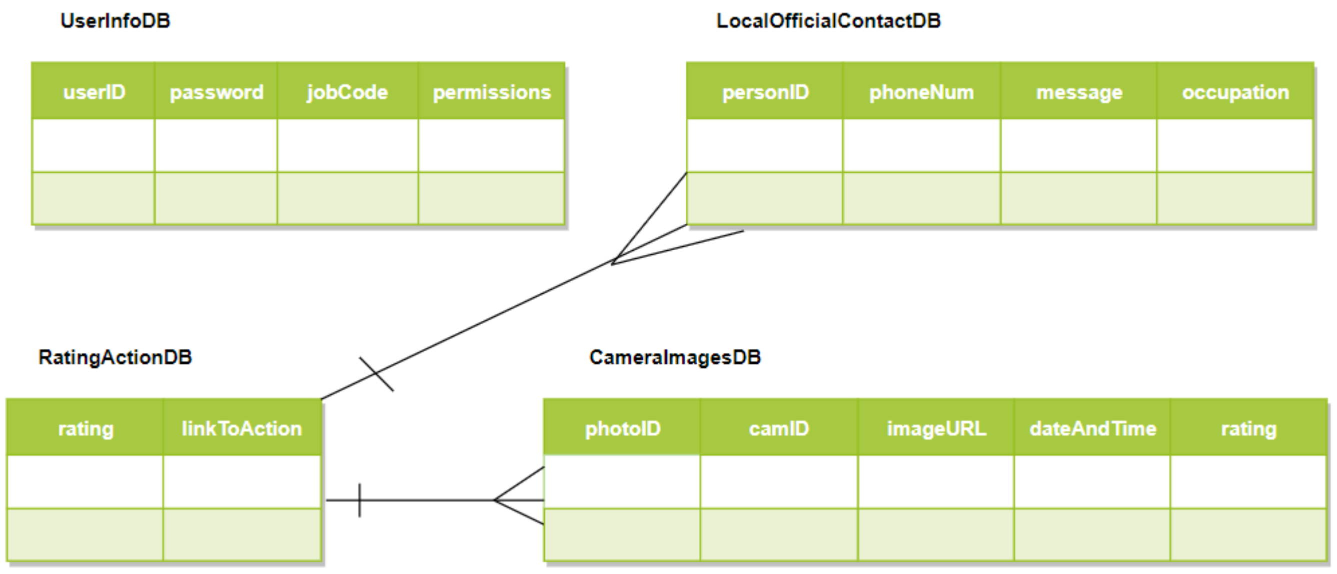
Software Architecture Diagram


Central Site Processing System
Comparison Algorithm
Holds photos (new and old), ratings, and camera accessing data
User Interface
Algorithm Interaction
Camera Controls
Rating Action
Login
Camera DB
Newer Photos <taken within a month>
Older Photos <up to a year ago>
Officials DB
List of Officials to be contacted in case of emergency <police, fire department, Amtrak, City>
Rating DB
List of Ratings with a link to action depending on the rating
User DB
User Credentials <Job Code, Permission>

Software Architecture Design Description

In this SWA diagram, a clear picture of how the system will be implemented is on display. Starting with the camera icon, this represents the six cameras placed along the cliffside, monitoring the change in stability. These cameras are then uploaded via Wifi to the nearby router, which transfers the data into the Camera Database. This database is what holds all the photos, new and old, and will be accessed by the Central Site Processing System. This system has a Comparison Algorithm implemented to extract the data from the Camera Database. This algorithm holds the photos, ratings, and camera accessing data, along with actually assigning the rating. The output of this algorithm is then sent to the Algorithm Interaction feature of the User Interface. From here, the user can see the photos with the rating attached to them. Depending on the rating, it is then sent to the Rating Action section of the system, where it will access the Officials Database, User Database, and the Rating Database. The Official and User Databases contain the contact information of important officials in the area that need to be contacted if there is going to be a slide, as well as verified users that are qualified to operate the computer controls. The Rating Database holds the different ratings an image could have and a link to action, so if an image has a certain rating then the Rating Action will use the database to perform the appropriate action. There are also camera controls that interact with the login feature, which accesses the User Database to see if the user who is attempting to log in has valid credentials to access the system. It is sort of daisy-chained in the sense that in order to access the camera control feature, it first needs to interact with the login feature, which needs to interact with the database. The reason why the Rating Interaction does not require login access is that if the rating is high enough, it should show a warning regardless if anyone is signed in. This prevents disaster from occurring because a user is having trouble logging in.

Data Management Strategy



This diagram represents the databases we need for our desired data management strategy of the bluff monitoring system and the relationships between these databases. We chose to make these all SQL databases because there shouldn't need to be any new columns added to them and each row needs to have all of the fields of information, so there won't be varying lengths of rows. Furthermore, accessing all pieces of data from a singular row all at once is a necessary action and we do not need more flexibility than the rigid, yet easily understandable, structure SQL provides. There will be a UserInfoDB (with DB standing for database in this section) in which each row will hold one qualified user's login and permissions information. The userID is an integer datatype and unique to each qualified user of the system. The password is a case sensitive string and the jobCode is an integer that is then tied to a permissions string. The next database is the LocalOfficialContactDB in which each row is a unique local official that has the possibility of being contacted when a linkToAction is enacted due to a certain image rating. The personID field is a unique integer identifier for each local official and the phoneNum field is of type double since an integer would not be able to hold values up to 999-999-9999 that is necessary to have all possible phone numbers accepted. The message field is a string that holds the personalized portion of the message that would be sent to that specific person if necessary that includes their name and responsibilities, though the specific actions are within the linkToAction based on the rating. The occupation field is of type integer and holds the individual's occupation code (similar to jobCode in UserInfoDB) that can be a determiner in whether they need to be contacted in the different emergency scenarios. Then, there is the RatingActionDB that is very small containing only five rows, each being a unique rating and its corresponding action. The rating field is of integer type and has only numbers 1-5 since those are the only possible ratings for images received from the comparison algorithm. The linkToAction field is a string that when enacted, will pull from the LocalOfficialContactDB in order to carry out the proper response to a certain bluff image's rating. Therefore the relationship between these databases is 1 rating action has many local official contacts, since a more severe rating (higher number) would require alerting every active local official in order to stop railway traffic, evacuate areas, and save people's lives. Finally, the last database is the CameraImagesDB in which each row is a unique photo of the bluffs that has been safely transmitted from the system's cameras. The photoID field is a unique integer that is attached to the image in order to identify it and have a chronological ordering to them since the id numbers are generated sequentially as new photos are added to the database. The camID field is an integer that specifies which camera took the stored image. The imageURL holds the image as a string in a URL link format that is received and/or sent to the comparison algorithm. The dateAndTime field is a pair of integers with the precise date and time that the image was transmitted from the camera to the central unit of the system. The rating field is an integer that is determined from the algorithm and remains attached to the image for future reference if necessary. The connection between the RatingActionDB and the CameraImagesDB is that many camera images can have the same singular rating action.

Trade off Discussions

One tradeoff we considered is that we could've split UserInfoDB into 2 tables, removing the permissions from the original database and adding a new much smaller one with only jobCode (to relate to the original) and permissions since they are often accessed together and job codes and permissions are often the same for many different users with similar qualifications. This would decrease the amount of space needed in the current UserInfoDB table, which is helpful since the database would only have to be accessed for logging in and then the other database would be accessed when finding out about user permissions, since they are separate actions done at separate times in the system progression. We chose to keep it as one database to simplify our diagram and because then we can give different permissions to people with the same jobCode depending on their training/knowledge level. Also, we chose to make all of our databases SQL tables for their rigid structure, since it is easier to understand visually and our system didn't really need any additional functionality than the limited relationships and abilities that SQL provides. On the other hand, we could have made CameraImagesDB into a NoSQL because then we could've used a structure that stores images in a more efficient and technologically appropriate way than simply as a URL. Though, if we did switch the structure, then it could be more problematic to attach the rest of the data to that image and when the routine cleaning out of certain old images occurs, there is likely to be more difficulty removing specific images from a less rigid structure than simply deleting a row in a SQL table to remove an image. A change we made from our old software architecture diagram was splitting up the user info and officials' contacts databases. We believe this was necessary since the data for users and local officials are never accessed at the same time and don't have any overlapping characteristics, so it would have been very inefficient to hold both of their data in the same database.