SDSU | San Diego State University

Kili Trekker Software Design Specification

Prepared by: Matteo Gristina, Ryan Jaber, Neel Raj

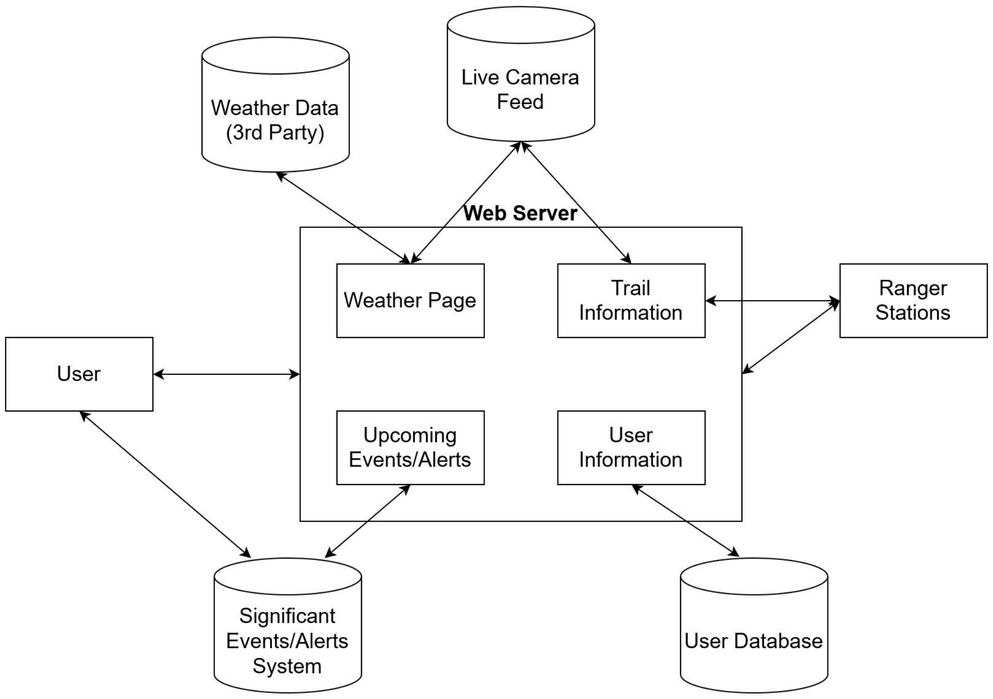
April 20, 2023

Table Of Contents

1 Software Architecture Overview: 2
1.1 Software Architecture Diagram 2
1.2 Software Architecture Description 2
2 Data Management Strategy 3
2.1 Data Management Diagram 3
2.2 Data Management Description 3
2.3 Data Management Relationships/Strategy 4
2.4 Data Management Alternatives and Trade Offs 4

1 Software Architecture Overview:

1.1 Software Architecture Diagram

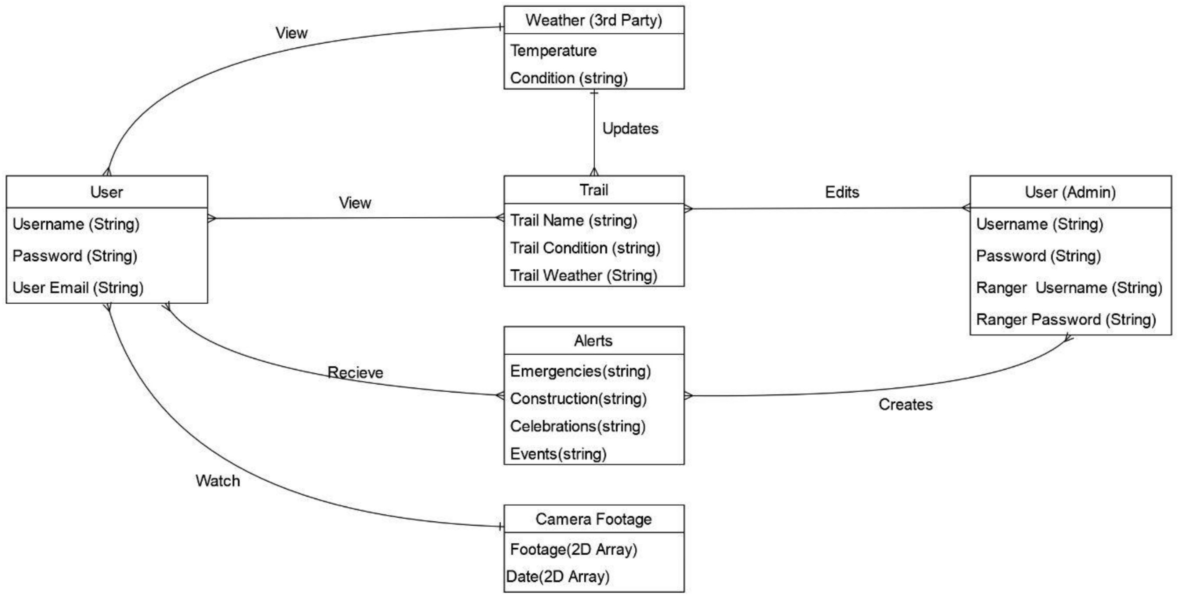


1.2 Software Architecture Description

The architecture for the Kili Trekker system is based on a simple web server/website, so it has a Client-Server Architecture. The user interacts with the main web server which is made up of 4 components: Weather, Trail Information, Upcoming Events, and User Information. The web server is accessible by the user in read format, the user cannot update any information. The weather page interacts in a two way fashion with 3rd party weather data and the live camera feed from the summit to accurately display the current and upcoming weather conditions. The server sends requests to both components and is sent back data to be displayed. The trail information cannot be automatically updated, so computers at ranger stations can be used to directly update information and trail conditions. The stations also have write access to the web system as a whole, as requested in the information given by the client. To keep track of users, the User information component reads and writes to an external database. Lastly, the upcoming events works together with the Alerts system to directly communicate with the User in case of emergency, or display non-emergency events occuring in the park. The User and Alert system form a Publish-Subscribe architecture, where those opted in to emergency notifications will receive them. The updates to the diagram include changing the components to databases to clearly show which elements of the system are going to handle data in the form of a database.

2 Data Management Strategy

2.1 Data Management Diagram



2.2 Data Management Description

To handle massive amounts of data and store such data is why we decided to go with NoSQL, because we believe that as the data begins to add up, the sql database will need more hardware components to increase the data volume. In addition, the data bases are following the CAP formation of Consistency, Availability, and Partition tolerance which is in correlation to how the Kili Tracker System needs to be in terms of data. The most important of the three being Availability, as the system needs to satisfy users from around the world. The trail and weather data must be consistent and available for our rangers and users. The system will have large data sets, for example camera footage or temperatures, and thus new data can be easily inserted since there will not need any prior steps. More so, we do not need correlating tables, we only need to display and document the data that the rangers deem necessary.

We need the data to be scaled at an acceptable cost since we will be in the mountains and trails of the park. Furthermore, the new database hardware has to be built over the previous

IT systems as the applications are replaced or added. If we scale out or horizontally scale, we can use other connected servers to run the necessary servers and add the servers as needed. This is another reason we chose NoSQL, as SQL is not horizontally scalable. Additionally, being able to run over multiple servers is a prime reason why we picked noSql over SQL since we have backup servers just in case of interruptions.

2.3 Data Management Relationships/Strategy

The diagram above shows the different databases and the interactions or relationships between two or more pieces of data. Firstly, the user is split into two different types of users, Normal and Admin. To be consistent with the requirements for the system, admins are granted read, and write access to the databases that are appropriate. This rules out the Weather and Camera footage databases, because weather is going to be collected by a third party, and the camera footage is just to back up the feed from the summit. This leaves the trail and Alert databases. Both have a many-to-many relationship with Admins, because Admins can edit multiple trails, and trails can be edited by various admins. The same goes for Alerts. On the other side of the diagram, Normal users have viewing permissions to the weather and trail databases. Users receive alerts in a many-to-many relationship, but Users can watch the camera footage in a many-to-one relationship. For the camera footage, there is only one camera at the summit, so a many-to-many relationship is not possible.

The overall structure of the system is based upon providing information and a user viewing it at any time, so the data management strategy reflects that. Only one party, the admins, edit data, and do so sparingly, while the rest of the relationships between are some form of viewing, or sending data. The data management diagram can be a little misleading, because as the Architecture diagram portrays, the user mainly interacts with the data through a web server, so we do not need to have individual copies of the data for each user. The largest database storage-wise is going to contain the user information/admin information, because each registered user will have their own set of data that our system will store. The Number of trails, alerts, volume of weather, and camera footage will not increase at the same rate as the number of users.

2.4 Data Management Alternatives and Trade Offs

We chose to use NoSQL because it made a lot more sense for our system. The main constraint with our system is that we had to complete it within a reasonable budget since the system itself will not be used by more than 20,000 people at a time as a rough estimate with plans to scale. Scalability would be a lot easier with NoSQL because we wanted the system to be able to handle multiple types of data and some of which would have no connection to another such as weather conditions and alerts for one.Also, since our system will not be handling any extreme tasks and mostly will just be a way to display data, NoSQL made more sense for that because the performance of a NoSQL database is faster than that of its counterpart.

While we decided to implement NoSQL, there are some things we are giving up by doing so. We have some datasets that link to each other that would be easier to display using SQL but this is a fairly small downside. Another thing we are giving up is the ability to do more than just display information if we ever decide in the future to implement more complex features and interactive tasks. We also lose the ease of use that comes with a SQL data management strategy which will require us to go into more extensive training for staff to be able to use and understand our system. Lastly, we lose the ability to use monitoring and restore tools that are inherent in a SQL strategy.

But, overall, the good outweighs the bad in terms of cost and ease of implementation for the short time for our software system. However, we may consider standardizing our data with SQL depending on what extra features we add in future implementations of our system.