

StormSpot

Overview

Motivation

Tornadoes are the most intense storms on the planet, with wind speeds that can exceed 300mph. The key to surviving these storms is to get advance warning and react. Since tornadoes are small compared to other storms, such as thunderstorms, they are more difficult to locate on weather radars. For precise tracking, meteorologists rely on citizens to spot storms and report them. These eyewitness reports help pinpoint a tornado's location.

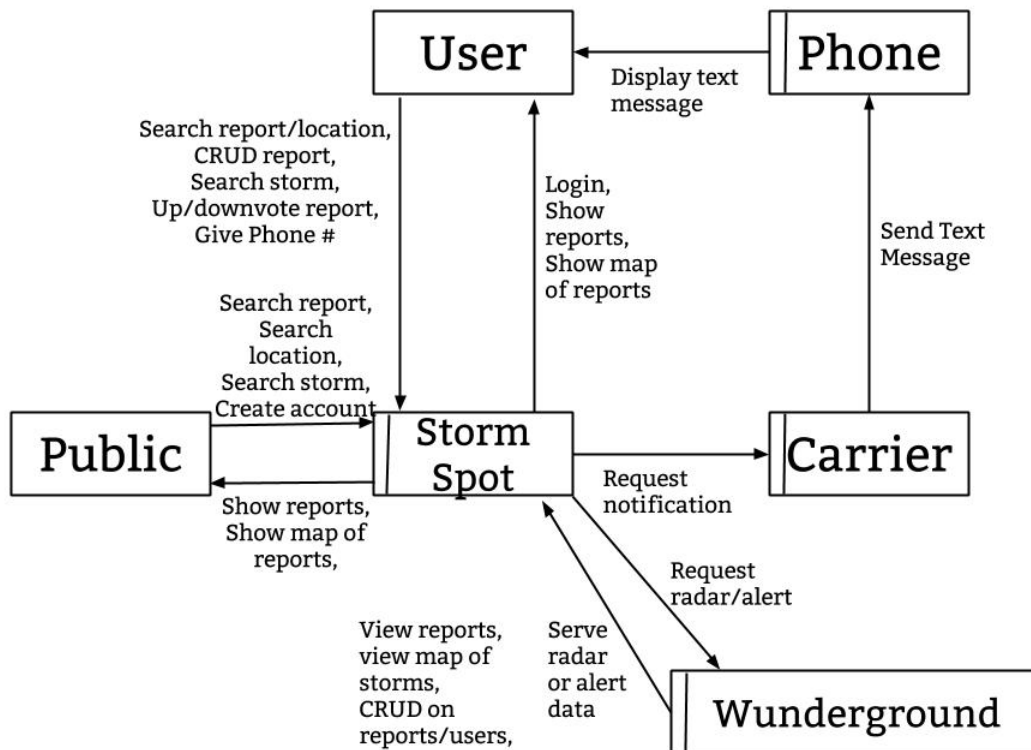
Purposes

StormSpot aims to solve the problem of inaccurate storm reports because the existing solution, the weather radar, does not achieve great enough accuracy. It will achieve this by aggregating storm reports from users.

The app has three main purposes in order of importance:

1. Show user where the storm is located.
 - a. The user should be able to see the locations of storms (ex. tornado, severe thunderstorm, hail, strong winds) with descriptions (ex. golfball size hailstones) and locations down to the street level. Users should also be able to see roughly where the storm is going.
2. Let the user report a storm.
 - a. When the user sees a weather event of interest, they can report it with a description and pictures. Other users of the app should be able to see the user's report. One potential problem is inaccurate or false reports – to avoid this, we shall allow users to upvote or downvote reports to argue in favor or against their authenticity.
3. Notify the user about storms near them.
 - a. When there is a storm near the user, they should receive a notification even if they aren't using the app. The notification should tell them what the storm is and where it is located.

Context Diagram



The context diagram is displayed above. There are three primary roles that interact with StormSpot.

The public (i.e. somebody who has not registered), can view the storm reports and a map indicating the storm reports.

A user (i.e. somebody who has registered for an account) can also view the reports and the map. However, a user can also create, read, update, and delete their reports. They can also upvote/downvote reports to corroborate (or testify against) their authenticity. Finally, a user can give their phone number so they will receive text messages when there is a storm near their location.

There are three systems that interact with StormSpot.

When StormSpot collects phone numbers from users, it aims to send text messages to the user to notify them about storms near their area. To achieve this, we ask the user to provide their cell phone carrier so that we can send text messages to them via email using the module nodemailer. Then, the user's phone (another system) displays the text message to the user. StormSpot cannot control the interaction between the carrier, the phone, and the user. Unfortunately, this means there is no way to guarantee that a text message was delivered to a user.

The third system is Weather Underground (Wunderground). Wunderground provides an API to retrieve alert and radar data. StormSpot retrieves this information and overlays it on the map displayed to the user.

Design Model

Concepts

There are three main concepts in StormSpot.

Report:

Brief Definition: User submitted information about a storm.

Motivating Purpose: "Let the user report a storm". This is the entity that captures the information that user reports about a storm.

Description: A report includes a timestamp (when the report was made), a location, an author, a number of storm types (ex. tornado, thunderstorm, wind, hail), a severity (the higher the severity, the more dangerous the storm), an authenticity (i.e. the number of users who upvoted the report minus the number of users who downvoted the report), a text description.

Storm Map:

Brief Definition: A map displaying trustworthy reports, alerts, and radar imagery.

Motivating Purpose: "Show user where the storm is located". By displaying a map overlaid with reports, severe weather alerts, and radar imagery, the user can identify the storm's precise location.

Description: The app will display a map overlaid with the trustworthy reports (i.e. the reports with the highest authenticity value) and optionally the severe weather alert data and weather radar imagery. By scanning the reports and looking at the alerts and weather radar imagery, the user can see precisely where the storm is located. By animating the reports over time, the user can see where the storm is moving.

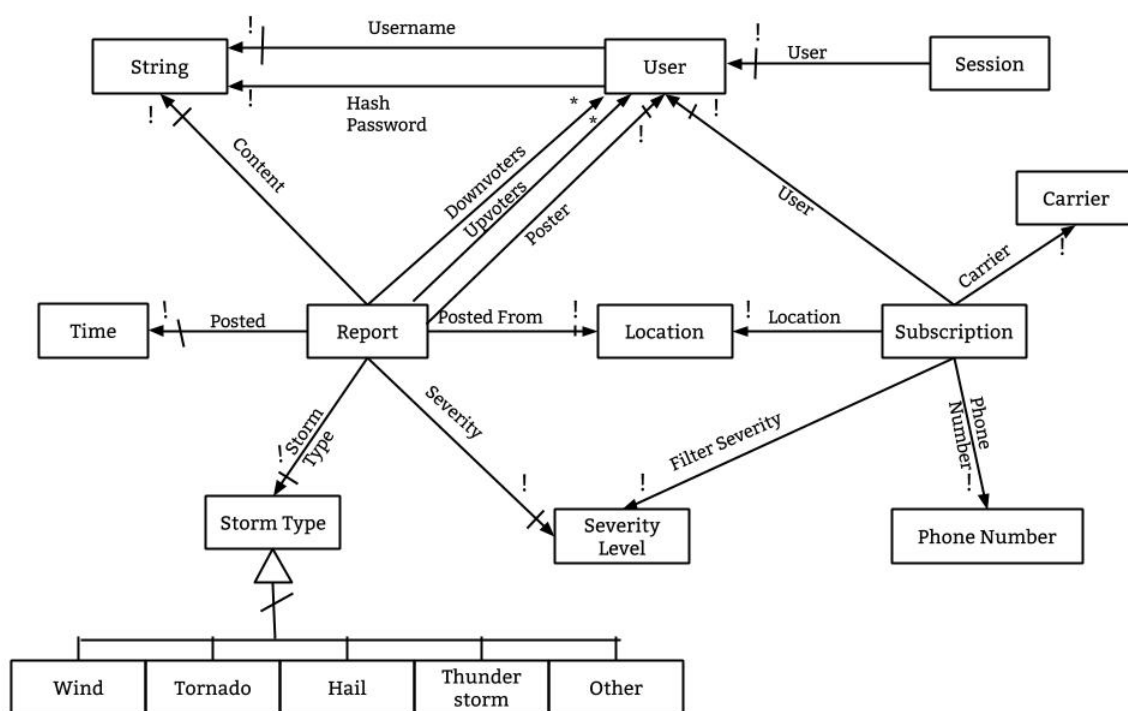
Subscription:

Brief Definition: A user's subscription to receiving text messages when storms are near them.

Motivating Purpose: “Notify the user about storms near them”. By subscribing to text message alerts when storm reports occur near him/her, a user will be notified when storms strike.

Description: A subscription consists of a phone number, a carrier, a location, and a severity level. The phone number will receive a text message when a storm report with a severity greater than or equal to the severity level is reported near the location.

Data Model



We capture the application in the data model above.

First, notice that if the public (i.e. a non-user) uses the app, there is no need to store their information in the database, so we do not incorporate the public into the data model.

A **Session** is associated with a single role that cannot change. Sessions prevent users from having to log in multiple times.

Users have immutable usernames and mutable passwords.

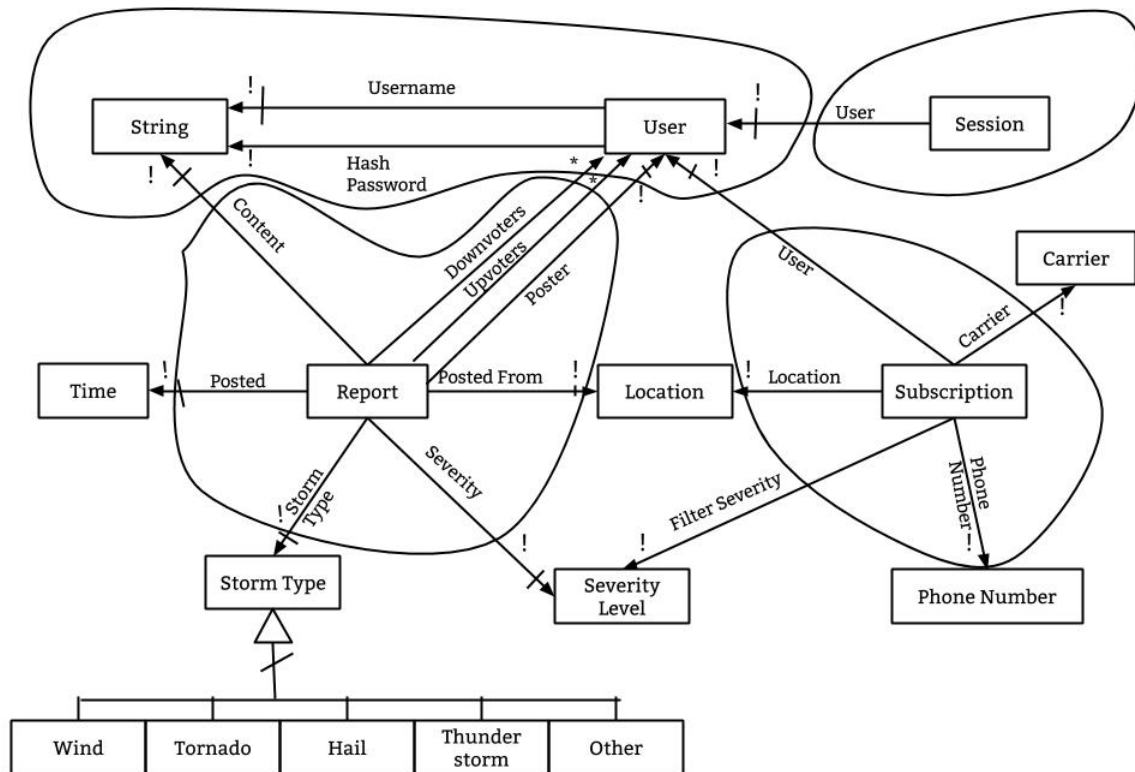
A Report has an author, post time, severity, reported from location, and some storm types, each of which cannot change. A report also has a set of up-voters and down-voters. We will use these sets in the future to measure the authenticity of a Report.

There are multiple storms that are subsets of Storm Type.

A Subscription is associated with one user, which cannot change. It also has a filter severity, monitored location, phone number, and a carrier each of which can change.

You will notice that the data model makes no reference to maps, weather alerts, or radar imagery. This is because that data is retrieved from other services (ex. Google Maps and Wunderground). Therefore, there is no need to store that information in our data model.

Data Design



We draw the contours above, which let us represent the data model using four MongoDB collections. They are:

Session

user: ObjectId pointing to a User

User

username: String
hash_password: String

Report

poster: ObjectId pointing to a User
posted: Date
posted_from: GeoJSON point
content: String
upvoters: Set(Users)
downvoters: Set(Users)
severity: String
storm_type: Array of distinct strings where each string is "Tornado", "Wind", "Hail", "Thunderstorm", or "Other"

Subscription

subscriber: ObjectId pointing to a User
phone_number: String
carrier: String
severity: Number
location: GeoJSON point

Challenges

Design Challenges

Challenge: How to we check the authenticity of a report

Options: Use the up-votes and down-votes to measure the authenticity; use all reports regardless of votes

Choice: Use the up-votes and down-votes to measure the authenticity

Justification: We don't want to be displaying false information on our map, but we need a way to check if a particular report is valid or not. So we decide to put this job into the hand of the users. By allowing users to up-vote or down-vote a storm, we give them this power. Now we just need to decide what is the maximum number of down votes that we will allow a report to have before we discard it indefinitely.

Challenge: What reports should we display on the map

Options: Show all reports; show most recent reports; show all reports received in the last 30 minutes

Choice: We decided to give the User the option to select what reports they want to see (i.e. Most recent reports, reports about a certain location, reports received over the last 30 minutes)

Justification: One of the main goals of our project is to allow users to be able to track storms and their movement. In order to do this, we want to have some concept of time associated with how we show the reports on the map. This way, we may

animate the movement of the storms by showing the reports overtime. We also do not want to restrict ourselves to this ability, so we give the users a choice seeing all reports.

Data Design Justification

We've separated our data into four collections. This way, we decouple information that is likely to be retrieved separately. It does not make sense to group the Subscriptions and the Reports together. Since we may seek to retrieve the reports independent of the User (ex. Get all reports where the storm type is a tornado, get all the reports near a given location), we segregate the Users from the Reports. Similarly, since we often query the Subscriptions independent of the Users (ex. get all the Subscriptions near the given area so we can send them text messages about the storm in the area), we segregate the Subscriptions from the Users. If we want to retrieve the Subscriptions or Reports for a given User, we can query by the subscriber or poster, respectively. Therefore, the data design is effective in decoupling our different kinds of data, but also quite efficient.