# 1. Scenario

Develop a “Bird Watching” progressive web application that provides users (typically bird watchers) with means to record and view Bird sightings and also to help with identification. You will need to develop this as a web application, with the supporting server infrastructure. Using the website, the users can add new sightings, view sightings and also comment on identification.

The app should allow access to a sorted list of bird sightings. The system will allow:
- Viewing sightings sorted (at least) by date/time seen, and whether identification is finished.

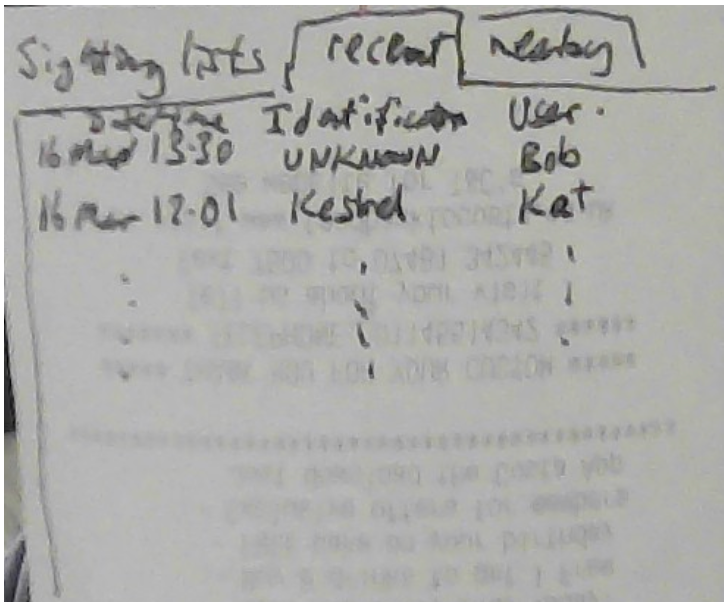
- The addition of a new sighting:

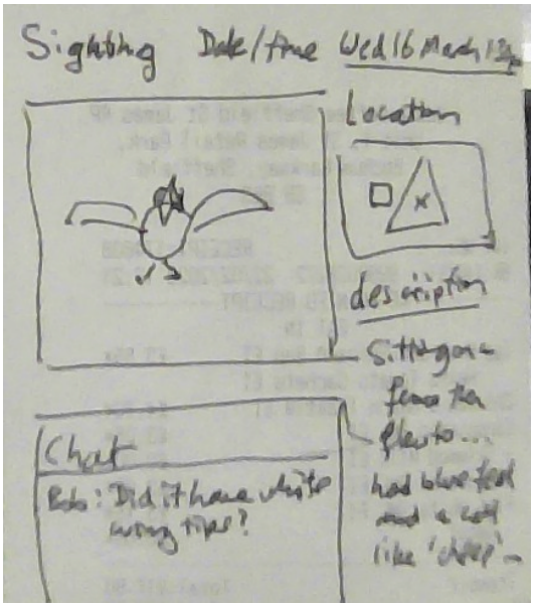
- Note that once added, most of the sighting will not be modifiable
- Each sighting contains (at least):
    - date and time seen
    - location (i.e. geolocation)
    - a (short) description of the sighting
    - an identification (which may be unknown or uncertain) - this can be updated:
        - The original user who sighted the bird needs to be happy with a suggestion and can update the sighting to show the identification
        - The identification should be linked to information obtained from the DBPedia knowledge graph. The information includes a common name/scientific name of the bird, an english language description and a URI (which should link to the DBPedia page describing the bird).
    - a photo (usually)
    - the user’s ‘nickname’; as a string - there is no need to implement a login system
- A chat/comment linked to each sighting which contains:
    - comments on the sighting, i.e. discussion and suggestions about what the bird is

When a new sighting is added, it is accessible to all users (i.e. you do not need to implement a login system or a set of privacy rules – when the user enters the site they will see all the sightings).

# 2. Design

In the sketches below you can see some initial ‘back of an envelope’ sketches of a list of sightings (left) and individual sighting detail (right). The design is left to you - please improve on the sketches below. Note that you will need to add/amend the detail on these sketches and also that you will need to add new sketches for (at least) adding a new sighting and updating the identification status. A good approach would be for the sighting detail interface to also work for adding new sightings and updating the identification.




When clicking on a sighting in the list, the individual details will be shown. For the original user who reported the sighting, this will allow them to update the 'identification' detail. The other details are fixed after creation, but it must be possible to chat (in real time) about the sighting. This chat is public to everyone who views the sighting. The original user might also use the chat to add more detail/s.

## 3. Components

The following is a list of components that must be included:

3.1. The Web Application

- The web application must:
  - be progressive
  - support online and offline interaction
  - be implemented using EJS and JavaScript in a node.JS framework
  - be non blocking
- The web application can
  - be multimodal (i.e. provide access via multiple devices, such as mobile phone, laptop etc).

FUNCTIONALITIES

- The web application must allow creating new sightings with associated details - description, date/time, user nickname, location (which should come from geolocation or optionally selecting from a map) and an associated image (photo)
- The web application must allow viewing all sightings with associated details - description, date/time, user nickname, location (which should come from geolocation or optionally selecting from a map) and an associated image (photo)

- Sightings can be sorted by most recent and (stretch goal) by distance away (as the 'crow flies')
- A sighting can be selected for more detail/chat by clicking on it.
- When online, the details must be sent to the database
    - Images likely need to be transformed to base 64. Images are uploaded to the server and stored in the MongoDB database. The images can be uploaded from the file system or can be referenced using a URL.

- When the user is offline, you must allow changes to be stored locally (for uploading later), including:
    - it must be possible to create (at least) one new sighting in the browser (indexedDB)
    - it is strongly preferred for more than one (new) sighting to be held offline in the browser
    - it must be possible to add chat messages to sightings the user has newly and previously created
    - N.B. You must not hold all sighting details offline, since only a selection will be relevant to each user. You should keep a local copy of the list of sightings, but not all the details.
- When the device is online again, you need to:
    - firstly upload local changes to the server - this should be a safe operation and should not need to consider any server changes
    - Note that after this, you should be able to safely reload from the server
    - retrieve updates (i.e since last synced) to the list of sightings (retrieving everything from scratch is a weaker solution). This should be new sightings and (only) changes to the identification of sightings.
    - you should also reload any chat messages related to the user's created sightings
    - As a stretch goal, you should notify the user of new messages in their chats (i.e. for sightings they created).

- The sighting creation can be implemented as a form where you provide text, a photo, etc.. The photo should be (at least) by uploading through an HTML5 form or (better) by taking a photo from the web camera - or phone camera (if you choose this option).

3.2. The Chat messages

- The chat part of the application must:
    - be progressive
    - support online and offline interaction
    - be implemented using socket.io in a node.JS framework
    - be non blocking
- The chat application can
    - be multimodal (i.e. provide access via multiple devices, such as mobile phone, laptop etc).

FUNCTIONALITIES

- When a user selects a sighting, the (live) chat must appear (likely at the bottom)

- Any new message must be visualised in real time, as it becomes available
- The user must be able to add new message/s which will appear in real time on other user's chats (when they have it open). As a stretch goal, users should be notified if someone has added a message to a sighting they created.
- When online, these should update in real time.
- When offline, messages can be added to sightings that the user created, and the chat will 'sync up' when online with new messages being shown and made visible to everyone viewing the sighting detail
- Since messages are linked to a specific sighting (eventually), you might choose to store the chat within the sighting document/object in MongoDB
- Note that messages cannot currently be edited or deleted

You will likely 'broadcast' (from the server) any new chat messages with a 'sighting id', so the web app can choose which messages it needs to update - i.e. where the user has a sighting open - and/or should be notified since they created the sighting.

3.3. The server

- The server must:
    - be implemented using NodeJS+Express
    - must support the chat system
    - must connect to MongoDB.

3.4. Data Storage and retrieval

- Data storage must be implemented using
    - MongoDB as network DB
    - indexedDB as browser storage
        - Note: Using cookies or local storage is not appropriate for this module
    - For the information linked from DBPedia
        - use fetch-sparql-endpoint module
    - retrieve the correct annotations from the DBPedia SPARQL endpoint.

# 4. External libraries

List of allowed libraries:

- CSS/Javascript: Bootstrap
- NPM Libraries: Passport, Express, node static, body parser, fetch, socket.io etc.
- All code and any libraries used in the labs and lectures (e.g. socket.io, Axios, etc.)

Examples of NOT allowed libraries:

- Angular, React, React Native, Vue - note that server/browser frameworks will not be allowed
- Any languages building on top of Javascript and e.g. that requires compilation

NOTES:

Add:

- Comments to the code.
- A README.md file clarifying any installation/running instruction in the <MainDirectory>. N.B. Installation and running should still be easy/simple

All code must be developed in HTML5/Javascript/CSS/Node.JS. It must be able to run the project without problems on a standard machine. It should not need special installation of any external library other than running npm install.