CS142 Project #8
Photo App Sprint

A new management team enamored with the Agile software development methodology has taken over your Photo App. For the next sprint they have set up a scrumboard of user stories that you as an engineer on the project need to choose work from. Each story is assigned a number of story points and the expectation for the sprint is you will need complete stories that total at least 12 story points.

Setup

You should have MongoDB and Node.js installed on your system. If not, follow the installation instructions now.

Like in Project #7, you start by making a copy of previous project's directory (`project7`) into a directory named `project8`. Into the `project8` directory extract the contents of this zip file. This zip file will add a file named `submission.txt` which contains a form you will fill in as part of the submission process. Do all of your work for this project in the `project8` directory.

As in Project #7 you will need MongoDB and your web server running.

Problem 1: Implement some stories (75 points)

Below is the scrumboard for the project. Each User Story links to the story and hints on what the feature should do. You should take care in choosing stories to do. Stories interact with each other and can either make it easier or harder to do them together. For example, if you choose to do the story about adding visibility controls to photos, the activity feed story should not return photos that the user should not be able to see.

Unlike previous assignments, you have freedom to make any changes you want including but not limited to changing the database schema, web server API, and ReactJS application components. You are allowed (and encouraged) to bring in any 3rd party software packages to help you.

The one restriction on your freedom to develop is that your photo app has to make it through the class submission mechanism and run smoothly for the course staff grading it. Also, when implementing stories, you should take care to make sure that your web application does not crash during usage/testing (e.g., the webServer.js program should not crash during operation). See the submission information in the later sections for more details.

Scrumboard		
#	**User Story**	**Story Points**
1	Extend user profile detail with usage	3
2	@mentions in comments	6
3	Visibility control on photos	4
4	Activity feed	4
5	Deleting Comments, Photos, and Users	4
6	Photo like votes	4
7	Favorite lists of photos for users	6
8	Sidebar list marks users with new activity	3
9	Tagging photos	10

Hints

By design User Stories are not unambiguous specifications of a feature. This means you as the engineer will need to fill in the detail by choosing something reasonable in the places that the story does not prescribe an approach to use. One guiding principle is to answer the question “What would the user of the application want the feature to do?”

Grading

In order to get full credit on the assignment you need to do stories totaling **at least 12 story points (which equals 75 assignment points)**. Any story points you do beyond 12 will be applied as if they were assignment points (i.e., the conversion rate for any story point past 12 is 1 story point = 1 assignment point). You can submit additional stories totaling up to an additional 18 story points. In other words you can get up to 18 points of extra credit by doing a total of 30 story points. **We will only include 5 stories in the grading computation**.

We plan to be stingy with the partial credit for partially implemented stories. A strategy that does fewer stories completely will fare better than a strategy that partially completes more stories.

Note that if you use `npm` to fetch any new packages, you should be sure to add it to the `package.json` file (using the `--save` option should do this for you). If you fetch something not using `npm`, have it in your `project8` directory so it will be picked up by the submission script and available to the course staff grading your assignment.

To help us grade this assignment you need to provide guidance on the stories you implemented. The project8 zip file downloaded a file `submission.txt`. The `submission.txt` contains a form for you to fill in that collects the information we need. This includes your story point calculations, the points they are worth, and the total story points you achieved.

In addition to the `submission.txt` file, you must submit a short video tour of your photo app and the stories you implemented. The video length should be no longer than 12 * k seconds, where k is the number of story points you completed. A typical video length is 2 minutes or less. This should be a simple screen recording with audio. On Mac OSX you can use Quicktime and on Windows you can use VLC to take screen recordings. **Please do not spend too much time creating the video. You will not be graded on production quality.**

Videos whose length is close to the maximum allowed will likely be too large for our submission process so we need you to provide us a link in the `submission.txt` file. Uploading the video to a video sharing website such as YouTube or Vimeo is an easy way of getting such a link. As part of the grading process, we were planning on sharing on the class discussion forum some of our favorite videos. Please indicate in the `submission.txt` file if we have your permission to share your video.

Style Points (5 points)

These points will be awarded if your problem solutions have proper MVC decomposition, follow the MEAN stack conventions, and **Eslint warning-free JavaScript. You will want to run `` `npm run lint` `` before submission.** In addition, your code and templates must be clean and readable, and your Web pages must be at least "reasonably nice" in appearance and convenience.

Note: When installing dependencies that are not from `npm` (e.g., from bower), please modify the `package.json` file so that the script command `'npm run lint'` also excludes the folder(s) containing your dependencies. That way, the JSHint command will only check your own JS files and will skip over any JS files corresponding to libraries/dependencies that you use (which may not pass the linter).

Submission checklist

You need to package your photo app in the submission so that the course staff will be able to run it without the app requiring additional files (other than those listed in package.json). Our submission program will copy your entire submission directory except for the node_modules subdirectories.
Make sure:

- Any software packages you use in the project is either listed in the package.json or present as a subdirectory (not node_modules/) of the project directory.

    - The command "`npm install --save <module_name>`" will fetch a module and add it to your package.json.
    - **Please run a sanity check prior to submission by deleting your node_modules/ folder and then typing `` `npm install` `` inside the project folder.** Afterwards, check to make sure that your web app still functions properly when you run it.
- The `loadDatabase.js` script exists and works to load a clean instance of MongoDB with the appropriate schemas and objects for your photo app to run.
    - You may have to modify this script file depending on what schemas you add or modify (e.g., before the database is reseeded, you may want to remove all objects from any new collections that your web app supports).
    - **Please test that this will work from a fresh state by restarting mongod and then recreating the db and reseeding the database by typing `` `node loadDatabase.js` ``.** Otherwise, we may not be able to run your application.
- The command "`node webServer.js`" starts your web server and connects to a MongoDB instance on the localhost at the standard port address. The URL http://localhost:3000/photo-share.html should start your app.
    - Make sure to thoroughly test both the web server and web application (e.g., to ensure that the web server doesn't crash while the web application is being used/tested)
- All other project files required for your web app to run are inside of the final version of the project folder that you end up submitting on the server.
    - This includes any supporting images that are linked to as a result of seeding the database.
- A filled-in `submission.txt` document is included in your project folder.
- The specified short video tour of your photo app is uploaded to YouTube, and a link to the video is included inside of `submission.txt`.
- The JSHint command states that your JS code passes the linter. See Style section for details.

Note these most of these bullet points should be true for your project #7 submission so you only really have to change these if your new work affects them (e.g. import software, add schema fields, etc.). It's especially important that you follow the items in bold above or else we may not be able to run your web application for grading. Basically, we should be able to run your web application after running `npm install`, `node loadDatabase.js` and `node webServer.js` (and, in a few cases, following some special instructions that you outline in your submission.txt file) using only the files that you submit to us.

Deliverables

Use the standard class submission mechanism to submit the entire application (everything in the `project8` directory). Please clean up your project directory before submitting, as described in the submission instructions.

Extend user profile detail with usage

Story points: 3

User Story:

When viewing the details of a user, the user detail page should include the following:

- The most recently uploaded photo. There should be small thumbnail image of the photo and the date it was uploaded.
- The photo of the user that has the most comments on it. There should be small thumbnail of the photo and the comments count.
- Clicking on either of these images should switch the view to the photo's detail view containing the photo and all its comments. (i.e., the photos view you have previously implemented). The exact view will depend on if you implemented a photo stepper or not.

Hints:

- You will have to write your own server API calls. Data processing should **NOT** go in the controller.

@mentions in comments

Story points: 6

User Story:

The user should be able to @mention users in the comments of photos and be able to see all the photos that @mention a particular user.

- Enhance the add comment support to allow @mentions of any existing user. In the comment text box there should be an intelligent way to select users (see hint).
- Users should not be able to @mention an invalid (non-existent) user, and an @mention for an invalid user should not crash the app.
- Extend the user detail view to show a list of all the photos that have comments that @mention the user, or display something reasonable if there are no @mentions. Each item in the @mention list should include:
    - A small thumbnail of the photo. Clicking on the photo should link to the *location* of the photo on the user's photo page
    - The photo owner's name. Clicking on the owner's name should link to the owner's user detail page.

Hints:

- React Mentions: https://github.com/signavio/react-mentions (React)

Note that there is a difference between when a @mention is selected and when a comment is submitted. You may want to think of a way to mark the @mention in the comment text on selection.

You will need to modify the database to store @mentions that are associated with a specific photo. It might be useful to have API calls that add a @mention to a photo and return all photos with a user @mentioned.

Visibility control of photos

Story points: 4

User Story:

When uploading a photo the user should be able to limit which users can see a photo and its comments. The feature should work as follows:

- When a user uploads a photo they should be given an *option* of specifying a list of users (a sharing list) that can see the photo. If no list is specified (i.e. the user declines the option to specify a list) then the photo should be visible by everyone.
- If a photo has a sharing list associated with it, only the owner of the photo and those on the list can see it. It should not be reflected in any photo counts displayed to the logged in user if that user is not on the sharing list.
- If the sharing list is empty, then only the owner of the photo can see the photo. Note that this is different from “no list is specified.”

Hints:

- You will need to modify your schema to store the relevant permissions for a photo.
- You will need to modify the queries that return photos to only return those that a user has permission to view (i.e. do not filter the photos in the Controller).

Activity feed

Story points: 4

User Story:

A user shall be able to see a list of activities that have happened on the photo sharing site. The feature must have the following requirements:

- A button on the toolbar labelled "Activities" should take the user to a view that shows the 5 most recent activities that have happened on the web site in reverse chronological order (most recent at the top). Note that the displayed activities need not be unique (e.g., there can be 5 photo upload activities).
- Each activity entry should show the date and time of the activity, name of the user that performed the activity, the activity type (i.e. Photo Upload, New Comment, etc.) along with some activity specific information as follows:
    - A photo upload - show a small thumbnail of the photo.
    - A comment added - show a small thumbnail of the photo and the author's name.
    - User registering - no additional info
    - User logging in - no additional info
    - User logging out - no additional info
- The activity display view should either update itself when new activities occur or provide a refresh button so the user can trigger an update.

Hints:

- You will need to extend the database schema to store records of the activities performed. This can be done in many ways but an approach of creating a new object to store each record maps well to MongoDB. Computing the activity history can be done with a query across the activity record objects.
- Note: you do not need to have entries in the feed for pre-existing comments and photos (in the seeded database).

Deleting Comments, Photos, and Users

Story points: 4

User Story:

A user should be able to delete any of the things that he or she owns on the site. This includes:

- Any of his/her photos
- Any comments that he/she has made (this includes comments made on other users' photos)
- The entire user account. If the user chooses to delete his/her account, there should be a final warning prompt, followed by the user being logged out and all the details of his/her account destroyed from the database. This means that, in addition to the User object being destroyed, you should also handle the deleting of all photos and comments created by this user. Essentially, we will treat the user as never having existed in the first place. Other things added for other stories implemented (e.g. activities, visibility, mentions, etc) should also be covered.

Hints:

- When a user deletes various objects, you need to consider any associated information that must also be deleted. This will be done on the server and you may have to do some research into how to properly destroy objects with associations.
- Make sure a user can't delete anything that he or she does not own.

Photo “like” votes

Story points: 4

User Story:

- Similar to Facebook’s like button, implement a like button for each photo to allow the user to like the photo. (Can be a button, icon, etc)
- Clicking on the like button makes the user like the photo (and visually changes the like button to an unlike button). Clicking the unlike button on a photo that has already been liked by the same user should unlike the photo.
- The number of likes of a photo should never be greater than the total number of users.
- If a user has already liked a photo, there should be some sort of visual indication on the page that the user has already liked the photo. This visual indication should be removed if the user unlikes the photo.
- Next to the like button, it should display the number of likes for that photo. This count should be updated (visually) immediately upon liking or unliking the photo.
- A user’s photos page should be sorted by the number of likes in descending order (most liked photos at the top). If photos have the same number of likes, sort by the timestamp in reverse chronological order (most recent first). This doesn’t need to change immediately when a user likes / unlikes a photo - you are allowed to wait until the page refreshes.

Hints:

- You will need to update the schema to include the likes associated with a photo and the users who created the likes.
- You will have to perform validation on the server to ensure that each like is valid. Also look into race conditions and determine if this could pose any potential issues here for a large-scale system.

Favorite list of photos for users

Story points: 6

User Story:

A logged-in user should be able to favorite photos and have a user-specific list of favorites appear on a new page (path: /favorites). That page should be dedicated to showing a list of favorited photos of that user. Users should be able to remove favorited photos from that list.
Users should be able to intuitively mark any photo as favorite by clicking on a button appearing next to any photo shown on any user's list of photos. While a photo is favorited:

- the button to favorite that photo on a user's list of photos should be disabled
- there should be some clear indicator that this photo is currently favorited by the user

On a new view component the user should be able to see a list of favorited photos, which must meet the following specifications:

- Each photo must be represented by a small thumbnail
- Clicking on the thumbnail should cause a modal window to open up displaying a larger version of the image (see hint below). It's **not** sufficient to just have the click direct the user to a new page containing only that photo.
- The modal should also contain a caption consisting of the date associated with the image
- There should be an intuitive way to remove a photo (e.g., clicking the upper right x on a thumbnail) from the list so that it is not part of the user's list of favorites any longer, even if the user refreshes the page and logs back in

Other notes:

- The list of favorited photos should be be accessible via the url path /favorites (!#/favorites or #/favorites)
- Conditionally display navigation UI in the sidenav or toolbar to the favorites page depending on whether the user is logged in
- This list of favorited photos must persist across multiple sessions: i.e., you must use the server-side database to keep track of this list and access this information to construct the list when users visit their favorites page

Hints:

React Hints

- You may find https://github.com/reactjs/react-modal or https://github.com/jossmac/react-images helpful.
- A significant challenge for this story will be getting the dependencies installed and configured correctly for your photo sharing app.
- Any `UNMET PEER DEPENDENCY <package-name>` during `npm install` means that you need to install <package-name> before installing your current package.

Sidebar list marks users with new activity

Story points: 3

User Story:

Extend the user list on the nav sidebar to show the last activity done by **each** user (not just the logged in user).
Activities are defined to be:

- posted a photo **thumbnail required **
- added a comment
- registered as a user
- user logged in
- user logged out

The display should update automatically.

Hints:

- We recommend reorganizing the real estate in the sidebar to clearly distinguish who the current user is, what their recent activity is, and then the friends list with all their recent activities.

Tagging photos

Story points: 10

User Story:

The user shall be able to tag people in photos. Here is a rough sketch of the way this new feature should behave:

- Users can *tag* a photo by selecting a rectangular region of the photo and identifying the person in the selected region.
- The rectangle is selected by dragging the mouse across the photo. As the user drags out the region the rectangle must stretch and shrink. The user must be able to drag in any direction, and the rectangle must remain visible after the drag is complete.
- If the user tries to extend the rectangle outside the range of the image, you must clip the rectangle to the boundaries of the image.
- If the user doesn't like the rectangle selected, they can select another to replace it.
- Once the user has selected a region of the photo, they should be able to select a name from registered users of the system and tag the user. The tag can be submitted and recorded in the database.
- There can be multiple tags for a single photo, each with its own rectangle and user name.
- The application must provide a mechanism for displaying a photo with all of its tags: the rectangle for each tag should be displayed over the photo, and when the mouse hovers over the rectangle the first and last names of the tagged user should be displayed.
- Clicking on a user name should switch the view to the user detail view of the tagged user.

Hints:

- **This is worth 10 story points since it is hard!** On the front end it requires either finding some area select module and getting it to work or building your own area select mechanism. We suggest you choose a different story unless you are excited by the challenge.
- Each photo should have tags associated with the relative position in the photo. This will allow you to properly display the tags using relative positioning.
- Test tagging photos while the page is scrolled to different spots to ensure the position calculations are not sensitive to the window scroll position
- Some 3rd party libraries may expect you to provide a width and/or height for the image.