

# IT350: Final Project

## USNA meets Social Networks

### Goals and Dreams

The goal of this project is to create a social networking website that enables the Navy community to share and communicate personal (or professional) updates with each other in real time. The Naval Academy has been chosen as the Navy's testbed. You have been chosen as its developers.

Twitter began as a tiny test project in a small company called Odeo in 2006. Now Lady Gaga and Justin Bieber use it daily alongside hundreds of millions of people. It's your turn to make the breakthrough website that sweeps the nation. Start with your peers at the Naval Academy. The goal of this project is to create a website that lets users post status updates (e.g., as in tweets on Twitter, or statuses on Facebook), and read updates from their friends. Ultimately, the goal is a website that draws the Naval Academy closer together through technology.

Teams of 3 or 4 are required for this project. See "Project Roles" below.

### Overall Requirements

Twitter has its own vision of how tweets should be entered, viewed, and updated. You may adopt their general approach, but you are encouraged to come up with your own innovative ideas! Below are the requirements that you absolutely must implement. Some are more vague than others. You are to make your own design decisions where they are not explicitly stated, and you should feel free to add other subpages that are not listed here.

1. Welcome Page. The first page that users see when they visit your site. It should make clear what the site's purpose is and provide appropriate links to the main sections of your site.
2. Contact Page. A page that lists basic contact information for the site designers, and explicitly lists all team members with a short bio about each member. Adding a picture of each person would be a nice (optional) touch!
3. Search Page. This page should contain a search form and little else. The form should allow the user to submit a keyword search into your website that returns a list of status updates containing the keyword. The returned list should show at least 10 matching statuses (if there are 10 or more that match!). The results page does not have to be the same URL as the search page, but it should appear after the user submits the form. The search should be case insensitive, and is an exact keyword match only. For extra credit, allow regular expressions or logical operators.
4. Signup Page. For new users, provide a page that lets them signup and create a personal member page for themselves. The signup should contain a form that submits the initial biographical information to appear on their *User Page* (see below). You must include the following fields: (1) username, (2) password, (3) full name, (4) USNA class year, (5) company number, and (6) biography textarea. The password should have to be entered

twice, and you must check that they entered the same password before submission. Show an error message if not *and block the form submission*. Further, you must include, at the bare minimum, 2 other attributes for the user to fill in about themselves. It is up to you to choose relevant and interesting factoids to collect. See Hint \*1 for help on how to store this information when they signup.

5. User Profile Page. Each user will have a “profile” page that is publicly visible to all users. This page will list the basic biographical information that they submit when they signed up to your site (all fields except the password). This page should also list the user’s most recent status updates in reversed order (most recent on top). Each status should be shown with the time of posting. It is up to you to decide how many status updates to show, but you must show at least 10 (most likely more). If they’ve posted 200 status updates, you obviously don’t want to list all of them. Finally, the user page should include a list of the user’s current friends (see below). Note that if a “profile” page is requested for a user that does not exist, your program should generate some default page saying something like “this user has deleted their profile”.
6. List Members Page. This page will list all current members of the site, with links to those members’ User Profile Pages. You should list all members (no matter how many for the purposes of this assignment).
7. Member Area. Most users will be “members” of the website: they will have their own public user profile page and can submit status updates. The following pages should be accessible to members after they login.
  - a. Login Mechanism. You need a login page that accepts a username and password, sets a cookie upon user verification, and gives access to the following member pages.
  - b. Update Profile. (**not for teams of 3**) Members should have a way to edit/update their personal biographical information. This will probably be almost identical to the signup form, but you instead load their current info into the form fields for them to edit. See Hint \*1.
  - c. Post a Status Update. There should be a page that gives users a form to easily post status updates. Twitter only allows a single text submission of 140 characters. You must set some character limit to avoid users abusing your site, but it does not have to be 140. You should also enhance this text status with two other form fields that the user can (optionally) include: (1) current location of the user as a string they can type in, and (2) mood of the user, selected from a short list of options. You may also come up with other optional fields! All features should be saved with the status update when the form is submitted. See Hint \*2 for implementation ideas.
  - d. Add Friends. On Twitter, this is called “Following” people. On Facebook, it is your “Friends”. You should include a link or button on each *User Page* that says something like “add as friend”. This link should either only be visible to users

who are currently logged in, or show an error message if a user clicks it and is not logged in. Clicking the link should add that User to the logged in user's friends list.

- e. Personalized Status Feed. Each member has a list of friends they have added. This *Personalized Feed* will display a single list of status updates from all of these friends, ordered by submission timestamp. The list should show the most recent 30 status updates from all of their friends (you can do more if you wish). Each status should show all of that status' information (e.g., user's name, the status, mood, etc.). Note that this list is ordered, and the list will interleave updates from different friends assuming they post at random times. See Hint \*3 for suggestions on implementing this. If you are familiar with Facebook, this is like your Home page. Same with Twitter, it is your personal feed.
8. Administrator Area. The administrator should have a small area that lets him/her manage the website. Every website attracts bad eggs, so we want to weed them out. The administrator pages should not be viewable unless the admin is logged in. You can handle the admin like a normal member, logging them in through the standard member login page. You should somehow set a flag indicating which members have administrator access. These admin pages should not be viewable by anyone except a logged-in administrator.
    - a. Manage Users. You should have a page that lists all of the site's users and allows the administrator to delete users who are up to no good. How to delete is up to you, but see Hint \*4.
    - b. Statistics Page. You must have a page that lists the following basic statistics about the site's usage: (1) number of members, (2) number of status updates, and (3) aggregate counts of the moods over all of the status updates. Feel free to experiment and add others!

### **Other requirements:**

- You must add at least one innovative feature to your system that was not specifically required.
- Design and human-computer interaction: keep that in mind that this is a real website. At least 10% of your grade will be based on how well you implemented the design and HCI principles we discussed in class.
- The server side functionality shall be implemented with Perl.
- Login information will be stored and checked against a file *users.txt*. You decide the format of the file, but it must contain at least the following, if not the entirety of their bio information:
  - **Username:** username

- **Password:** the user must enter the correct username and password in order to gain access to protected functionality
- **User role:** basic member or administrator
- **First and last name:** user's real name
- Before accessing the Member area section, you must first direct users to a login page where they provide their username and password, which is checked against the users.txt file. Only after successful login they should be able to access the members area. You must use a cookie or some other mechanism so that:
  - a) they don't have to enter their username and password again
  - b) check that the user must have performed login and has the right role before accessing any information (e.g., admin-only section)
- Each member's status updates must be stored in a single text file for that member. You must decide the format of the file containing the information. We highly suggest you store a single status update *and all status attributes (such as mood)* on a single line, with fields separated by tabs.
- User profile pages and a member's personalized status feed must be dynamically generated based on reading in the text file (or multiple text files) that contains the status updates.
- Your site must be well-documented! You will be required to provide a brief overall write-up of how your site works. In addition, each file should have a brief comment at top on what it does, and comments within the code.
- You should define functions and share code appropriately. Don't have the same code in multiple files!
- You must make use of CSS to make your pages attractive and consistent looking.
- You must use JavaScript to provide useful checking of forms before they are submitted to the server. The Perl program should replicate those checks, to ensure users did not bypass JavaScript.
- A final report and presentation will be required. More details to be provided later.
- All **HTML web pages** must be constructed using Notepad++ or a similar text-only editor. The use of programs such as Microsoft Word, Microsoft Frontpage, DreamWeaver, ColdFusion, Mozilla Composer, etc. will be considered an honor offense.

## IMPLEMENTATION HINTS

\*1 *Storing/Editing biographical info.* You should have a single *users.txt* file that lists all current members, one per line. The line should contain the username, password, company, etc. When they edit their information, you will read in the file, and then overwrite the file again, but with the updated user information. How to store a user's information on a single line? Use a unique character to split the fields. Major companies like Google routinely store this sort of data, called feature/value pairs, separated by tabs. For instance, here's one user:

```
Username happyMID    first    Joe    last    Smitty    company    20    bio    I am originally from....
```

\*2 *Storing Status Updates:* you may want to create a separate status file for each member. Append a new status update to the end of the file on a single line. This allows for quick reading later, one per line. The status updates will have other attributes (time, mood, location, etc.) that should be stored on the same line with the status text itself. Use tab separators just as in the user file above, here's a possible format:

```
status    writing hints for IT350!!    time    2129:10212011    mood    happy    location    home
```

\*3 *Personalized Page:* There are several ways to implement this sorted status feed. Keep in mind that you will have to retrieve the status updates from all of a member's friends, and merge them together in order by timestamp. You may want to read each friend's status file one by one, and then append them into one large array. Perl provides a `sort()` function for arrays in which you provide your own custom sort comparator (similar to JavaScript). You'll want to sort by timestamp. Create a comparison function that takes two full status updates, where a status update is simply the string line you read from your status file (see Hint \*2). Have the function pull out the timestamp from each string, and then compare them. Return the appropriate value based on their order. The sort function will then give you the desired ordering. Display the first 30 (or more) status updates.

\*4 *Manage Users Page:* If you are storing all of the users in a *users.txt*, deleting a user can simply be removing that line from the file. You should also delete their status file and friends file, if you have them (you can use `unlink` function in Perl to delete a file).

## Project Roles

Each of you will have leadership responsibility for some part of the project. This doesn't mean you must do all the work for that part or only work solely on that part; it just means you are responsible for making sure it is done right. Part of your individual grade will depend on how well you lead your team in this regard. The roles are:

- Team Lead – you are in charge of ensuring that work is distributed in a fair manner to each team member and that the work is getting done. You are the primary point of

contact with the instructor (though anyone can ask for help) and should ensure that project reports and presentations are prepared as needed. The Team Lead should either be the one to apply changes to your shared web space, or else delegate this to someone else.

- Appearance Manager – you ensure that an appropriate stylesheet is created and used to create a consistent and attractive look across the whole site. Ensure that usability testing is done, and the design and human-interaction principles discussed in class are properly applied.
- Quality Control Manager – responsible for ensuring the system is functionally efficient, error free, and secure. The Quality Control Manager tests the system to expose its weaknesses and then interfaces with other team members to devise a solution. Additionally, quality controllers must ensure code is well commented and that similar tasks utilize the same code.
- (Not for teams of 3 people) Interface Manager – ensures that different pieces of the site can work together via well-defined interfaces. For example, the syntax of files containing status updates, as well as feedback information needs to be well defined (scripts that read and write the files must agree on what these look like).

## Tips

- **Start early!** This is a big project and you can expect to experience some unexpected delays or problems that may require changes.
- You want to be able to work independently, but as soon as possible try to interface your code with others. You are very likely to encounter unexpected issues due to different assumptions about how that interface should work!
- Make regular backups of your files!!! Especially when you have something working, make a backup in case it breaks later and you don't know why. Do this for both those files you specifically write and your site as a whole.
- Do most of your testing and development using your old W: drive. Then copy things to a team directory that you are ready to share with others. But be sure to communicate and have a plan so you don't overwrite things that someone else has done! The Team Lead should coordinate this.
- You will be writing a lot of Perl code to generate HTML. Make use of the HTML helper functions like `h1()`, `td()`, etc.

- If your code is not working, first run your program from the command line (like you did in Lab08) to ensure there are no syntax errors. Then, add extra print() commands to see what parts of the program are executing and what the values being used are.

## Extra Credit

Your project must meet the basic requirements, with the extra "innovative" feature. Then do any of the following for additional credit. You may also propose other ideas; talk to your instructor.

1. Create a poster that showcases the course and/or the project, and can be used as advertisement for the IT major. If multiple posters are created, we will vote on the best poster, and the winning team will receive an extra-extra credit. More information about how you can create a poster is available at <http://www.usna.edu/Users/cs/needham/ProjectPosters/ProjectPosters.htm>
2. Add a photo upload option to user's profiles. Even more Extra Credit: create a thumbnail sized version upon upload, and show with each status post on member's feeds.
3. Implement regular expression and/or logical (AND, OR) search options to the keyword status search.
4. Add a "delete status" option where the user can go back and delete any status they have posted. This will involve editing your current status file for that user, removing the single status and preserving all others.
5. Make "Company Specific Feeds" that show a page for a single company, and lists all of the status updates from all members in that company. It is analogous to a single member's *Personalized Status Feed*, but instead of showing a member's friends, you show all the members in a company. This can be a separate section of the site where you click on each company, bringing up their latest list of status updates.
6. Make your status feed webpage efficient! Webserver CPU time is expensive, and reading in all friends' updates into an array and then sorting is costly:  $O(n \log n)$ . There is an  $O(n)$  algorithm that manages the most recent 30 updates dynamically as you read in the status updates from disk. Implement it.

## Project Grading

A portion of your individual grade will be a group grade based on the following

1. Meeting projecting milestone and status reports
2. Functionality and correctness – The site should operate as advertised without error.
3. Visual Appearance & Consistency – including good use of CSS.

4. Maintenance – Ensure the site is well organized, requires minimal maintenance, and can be expanded **by someone else who doesn't know your code**.
5. Documentation – Ensure all pages are documented. Each file should start with a description of that file, creation date, and author name. If multiple team members modify the file, each should document the date and author of sub-sequent modifications. Each Perl function should be preceded by a short description. General documentation covering the system's layout and design is also required.
6. Creativity – The requirements above require at least one system innovation not specified in the project writeup. This and any other innovations will count towards your creativity grade.

The other portion of your individual grade will be administered by your teammates. Your assessment of your peers shall be based on effort, knowledge, team work, and professionalism.

## Deliverables

All items due at start of class unless otherwise noted.

- **Friday Oct 28.** Email to instructor: team name, role of each student, and what section of the website is each member responsible for (functional responsibility). When dividing the work amongst yourselves, remember that each team member is required to write at least one server-side script (Perl).
- **Tuesday Nov 8.** In lab, demo to your instructor that you have basic add status update working with the full form submission, and Perl write to file. No login is needed for the demo.
- **Tuesday Nov 15.** In lab, demo to your instructor that you can add a status, view a User Profile that lists all past status posts, and you can add a friend. It doesn't have to look good or have all features working, but you need to have all these parts working together.
- **Tuesday Nov 22.** In lab, demo to your instructor an end-to-end system: you can add a status, view a User Profile that lists all past status posts, and view a member's personalized status feed. It doesn't have to look good or have all features working, but you need to have all these parts working together.
- **Wednesday Nov 30 Projects and Reports due,** presentations in class.

**Presentation:** The members of your group should be prepared to demonstrate your system to the class. Be sure to mention your required innovative feature and any extra credit. You're not expected to prepare Powerpoint slides but if you wish place them in your team web directory for easy access.

**Report:** You will submit a single-spaced written project report, which includes

- A cover sheet with your team names and project members. Describe your innovative feature, any extra credit, and any requirements that you did not meet.
- A listing of the “role” that each person played (from the assigned set of 4) AND a description of what each person did – who created each file, who tested?
- Username and password for an admin user, and at least 3 other non-admin member users on your site. We may post status updates to these accounts when we test and grade your site.
- Half a page description of how you applied the design, human-computer interaction, and accessibility principles we discussed in class. Include description of affordances that you provided as well as feedback to user actions.
- A technical report of how the overall system works from a coding standpoint. We want to know how you use files, server-side scripts, etc. This should be no more than a few pages.
- All of your files should also be documented, also described in the requirements above.

**Feedback and Peer Grading:** You will be provided with an online form for giving feedback and grading the members of your team. You will submit this individually – peer grades are confidential. Check your email for more info.

## Late Policy

For each intermediate deadlines, there is a -2% (of overall grade) penalty for each business day being late. The complete project has to be submitted on time. No late submission accepted for the final project.

## Additional Hints/Clarifications (updated as the project progresses)

- **You should declare every variable by using 'my'.** I promise that this will spare you pain later!
- **How to connect to your shared T: drive – TBA**
- **Don't use the die() function.** This makes it hard to debug because the error message won't appear when you run through the webserver. Example:  
 NO: `if ($error) { die "Error in parse routine"; }`  
 YES: `if ($error) { print "Error in parse routine"; exit (1); }`

- **Hints on parsing input:** Once you've read in a line from a file, you will need to split it up into its different parts. A very useful function for this (and other parsing tasks) is `split()`. This function splits a given string into multiple parts whenever it sees a match from some pattern you give it. The results are stored in an array. Here are some example uses:

```
@myArray = split (":", $aLine); # (splits the line wherever there is a colon)
```

```
@myArray = split ("\t", $aLine); # (splits the line wherever there is a tab character.)
```

```
@myArray = split (/\s*\|s*/, $aLine); # (split the line wherever we see a vertical bar. Here the pattern also matches any whitespace (\s*) before or after the vertical bar, so that whitespace doesn't show up in the resulting array. This can be very useful!)
```

- **Don't forget to escape special characters.** A number of characters like `*` `|` `\` etc. have special meaning inside a pattern match. If you want to test for them, you'll need to escape them using a backslash as was done in the hint on parsing input above.
- **Plan so you can debug using the command line perl.** It's much easier to debug if you can run your program through the command line. You definitely want to do this for checking syntax but where possible you also want to do this for actually executing the program `./myPerl.pl`. For scripts that accept CGI parameter input, you can debug by explicitly passing in values on the command like this:

```
./statistics.pl "category=reporter"
```

- **Get the current time.** The function `time()` returns the number of seconds since Jan 1, 1970. You can store this when a status update is submitted, and use it to sort the updates later. You can convert these seconds into more human-readable dates and times using the `localtime()` function:

```
use Time::localtime;
my $timeobj = localtime($inseconds);
my $year = $timeobj->year + 1900;
my $month = $timeobj->mon;
my $day = $timeobj->mday;
print "The status was submitted on $day - $month - $year";
```

- **How to redirect in Perl:** In your Perl script, wherever you would want to use redirect, use the following statement (**without `print header()` or `print start_html()`...** before it):

```
print redirect("myNewURLHere");
```

`myNewURLHere` can be either absolute, for example `"http://MyWebsiteNameHere/myscript.pl"` or relative (to the current directory), for example `"myscript.pl"`.

- **How to store textarea strings:** If you want to store in a file the text received from the textarea of some user-submitted form, you might want to replace the new line characters in the text with some other character, such that your text fits in one line. One way of doing that in Perl is using the search-and-replace regular expressions. For example:

```
$textToStore =~ s/\r*\n+/&line;/g ;
```

will replace the end of line characters (either `\n` or `\r\n` depending of the system) with the string “&line;”. The above pattern will also get rid of multiple empty lines.

When you need to display the textarea string back to the user in a webpage, you can search for the “special” string and replace it with the end of line character. For example:

```
$textToDisplay =~ s/&line;/\n/g;
```

- **How to create a directory from Perl:**

```
my $newDir = "testdirectory"; #or whatever name you want here
```

```
my $perm = 755; #these sets the permissions; I would double-check that permissions are set correctly
```

```
if (-e "$newDir") { print p("Directory $newDir already exists") } # Checks for existing directory
```

```
mkdir ($newDir, $perm) || print p("Error making Directory $newDir"); #creates the directory or prints an error message
```

- **How to list the contents of a directory:**

```
my $dir = "testdirectory"; #relative path to the directory you want to list
```

```
opendir DIR, $dir || print p("Cannot open directory $dir");
```

```
my @fileNames = readdir DIR; #content of directory is returned as an array;
```

```
closedir DIR;
```

```
#print the content of directory, just to test the functionality
```

```
my $fileName;
```

```
foreach $fileName (@fileNames) { print p("$fileName"); }
```

- **How to upload files to the server:** The article at <http://articles.sitepoint.com/article/uploading-files-cgi-perl> provides a nice tutorial on how to create an interface for users to upload files (photos, etc) to the server.