EECS 485 Project 2: Authentication and Sessions

Due 9pm Saturday, Feb 6, 2016

In this project, you will continue working on the photo album website developed in project 1. However, do not touch the files in the pa1 sub-directory. Make another sub-directory called pa2, and copy the files from pa1 into the pa2 sub-directory and work on the files there. Modifying your pa1 folder after the project deadline is in violation of the honor code. Also, it can hurt the grading process. By the end of this programming assignment you will learn how to authenticate users and maintain sessions. Also, we are using new databases for this project, databases name: groupXXpa2.

Finally, make sure to change your routes to '{secret-key}/*pa2*/'. In general, we recommend using the blueprint argument url_prefix for specifying these routes.

Part 1: Getting started
Part 2: Build the website
Part 3: Authentication
Part 4: Validation
Grading

Part 1: Getting started

These sites contain useful tutorial and reference information for what you'll be implementing in this project.

- HTTP Cookies
- Sessions: Python
- Authentication: Python

Part 2: Build the website

This project is about personalization. The first step to doing any kind of personalization is to keep track of who is browsing your site. In class we discussed how HTTP is a stateless protocol, which cannot itself retain data from one request to the next. The way to maintain state from one page request to another is using **sessions**. In this project we will add a login page to the site. Users will only need to type their username and password once. Thereafter, your website will use session variables to determine who the logged in user is.

Pages that are *sensitive* require users to login before they can view those pages. The rest of the pages will remain public and will not require a username or password to be viewed. whenever a user tries to enter a sensitive page you should make sure that he/she has the privileges to view it. This is done by checking if the user has a valid session, and if so, whether the user is authorized to view the page.

Some pages do not require user authentication or sessions (e.g., a default home page or a create account page). Some other pages only require that the user be authenticated and are not dependent on who the user is (e.g., a logged in home page). Others may provide access depending on who the user is and whether he/she is permitted to access that page (e.g., someone's private album).

In short, when a user requests a URL, you should:

- Check if the page is public. If so, view the page.
- Check if the page is sensitive and the user has a valid session. If so, check if the user has permission to see the page, and if so, let them see the page. If they don't have access to the page, return a 403 error.
- Check if the page is sensitive and there is no session. If so, redirect to the login page.

Your code should observe the following rules about access privileges:

- Public albums are accessible to both logged in users and unauthenticated visitors.
- Private albums are accessible only to those users that have explicit access to that album (and the owner of the album). Users will have access to other user's private album if and only if there exists a tuple `(albumid, username)` in the `AlbumAccess` table (see below).

For example, User Matt creates albums Alpha (public), Bravo (private), and Charlie (private). Matt grants User Prateek access to Bravo. Prateek creates album Delta.

Prateek can view albums Alpha, Bravo, and Delta on his logged-in index page. These are the albums Prateek has permission to see.

Prateek can view **only** album D on his logged-in `/albums` page, by clicking on the link to 'My Albums'. These are the albums Prateek has permission to **edit.**

Matt has permission to view the albums Alpha, Bravo, and Charlie, both on his logged-in index page, and `/albums` page. He can edit these albums. However, he cannot view Delta.

Add public/private feature to the albums

Update Album Table

You need to add an `access` attribute to this table, so the new scheme for `Album` will be

- `Album(` **`albumid`**`, title, created, lastupdated, username, access)`

`access` specifies whether access to the album should be limited to a set of users indicated in the `AlbumAccess` table (described below). It only takes values of `public` or `private`. It should be an enum. More information about MySQL enums can be found here.

Create new Table `AlbumAccess`

- `AlbumAccess (`**`albumid, username`**`)`

This relation indicates the users who have access to each specific album.

In `/album/edit`, the user should be able to edit the `access` permission for the `Album` table. The user should also be able to give/remove user's access by editing the `AlbumAccess` table. Thus an album could be public, private, or private with someone accessible. Only the album's owner should be able to modify the album user's access permissions. If the album is not public, on `/album/edit`, you should have a table *that is only displayed for the owner* where the album's owner can view permissions of everyone with access to the album and type in a username to grant permissions. Public albums should not display this table.

The interface for `/album/edit` should appear roughly as below:

```
<table>
      <tr><td>Username</td><td>Update Access</td></tr>
      <tr><td>sportslover</td><td>[Revoke]</td>
      <tr><td>traveler</td><td>[Revoke]</td></tr>
      <tr><td>New: _____________</td><td>[Add]</td></tr>
</table>
```

The interface for `/albums/edit` should appear roughly as below:

```
<table>
      <tr>
            <td>Album</td>
            <td>Access</td>
            <td></td>
            <td></td>
      </tr>
      <tr>
```

```
            <td>Summer 2011 in Iceland</td>
            <td>Public</td>
            <td>[Edit]</td>
            <td>[Delete]</td>
      </tr>
      <tr>
            <td>Spring break 2010 in Brooklyn</td>
            <td>Public</td>
            <td>[Edit]</td>
            <td>[Delete]</td>
      </tr>
      <tr>
            <td>Thanksgiving 2010</td>
            <td>Public</td>
            <td>[Edit]</td>
            <td>[Delete]</td>
      </tr>
      <tr>
            <td>New: _____________</td>
            <td>________</td>
            <td>[Add]</td>
            <td></td>
      </tr>
</table>
```

The owner of an album should **not** appear in `AlbumAccess` for that album

Final Schema

The Final Schema should be the **same** as Project 1, except for the following additions:

1. Update album table to have attribute `access`
2. Add `AlbumAccess` table, as described above
3. Larger size for password (`varchar(256)`) to accommodate hashing and salting (see `/user` below).

Just like Project 1, you need a sql folder with the files `tbl_create.sql` and `load_data.sql`. You can/should use your Project 1 files as a starting point.

Session Management

Sessions allow us to reliably keep track of User information while the User is browsing our website. **This project is centered around the use of sessions.**

A good, concise explanation on sessions:
http://stackoverflow.com/questions/3804209/what-are-sessions-how-do-they-work

You may want to maintain one session variables: `username`. The `username` stores the username of the authenticated user.

Sessions in Python:

In Flask sessions are started automatically. The session variables can be accessed as such:

```
session['username']
```

In order for sessions to work properly however a secret key must be set. Please refer to the Flask docs for more information about how to use sessions (you will want to put the key in your app.py file). Be sure that sessions are imported when attempting to use them. Sessions work similarly in other Python frameworks.

The User's session is only maintained and used on the server-side; There is no need to write client-side code to maintain sessions. For simplicity in this project, assume cookies are **always enabled**.

Implement Sessions and Authentication:

What follows is a list of the url endpoints that you should create in your application. You should have created some of these for PA1. A **page is either in a public state (no session involved) or sensitive state (need session to manage page).**

Default home page: `/` [public]

Contains a welcoming message and information about the website. Also has an invitation for new users to join as members. There should be a link to a New User page. There should be links from this home page to all the public albums of all users. Sensitive homepage defined below.

Logged in homepage: `/` [sensitive]

This page is the home page for a user who has already logged in. Make sure for all pages in a logged in state, you clearly display the message "Logged in as `<firstname> <lastname>`" (in the header). This page and all subsequent logged-in pages should have a navigational interface with links to Home (this page), Edit Account (`/user/edit`), My Albums (`/albums`) and Logout (`/logout`). The main body of the page should have a list of all the accessible albums. Accessible albums include public albums, your own private albums, and private albums which you have been granted access to by the owner.

New User page: `/user` [public]

This page is for user creation. It contains a form that should `POST` to `/user` with `username`, `firstname`, `lastname`, `password`, and `email`.

Make sure the password field does not display uncensored text and that there are two password fields for verification. There are validation rules set forth in Part 4 below that describe the set of permissible passwords. You should use server-side validation for all data validation, including checking for duplicate usernames, checking that passwords match, and enforcing rules for valid names and usernames. Follow the validation rules in the section later in this document.

If a User provides all valid information, create the User by inserting their information into your database. Store the password like this: `<algorithm>$<salt>$<hash>`, where <algorithm> is the name of the hash algorithm ("sha3_512"), <salt> is a 64-bit hex uuid, and <hash> is the encrypted password. '$' is a literal dollar symbol. Here is an explanation on why we salt and hash passwords.

Here is an example of password encryption. Note: you may need to install the python package pysha3.

```
import hashlib
import sha3
import uuid

algorithm = 'sha3_512'   # name of the algorithm to use for encryption
password = 'bob1pass'    # unencrypted password
salt = uuid.uuid4().hex  # salt as a hex string for storage in db

m = hashlib.new(algorithm)
m.update(salt + password)
password_hash = m.hexdigest()

print "$".join([algorithm,salt,password_hash])
```

```
sha3_512$e5ee4ada47304adcaa683d97ad8ae56b$621a0903c98c80dc1f689ec86440
fe38cdd7b8237dbc50d481071f44827612433444cf330cbdeabdc852083cc6ed157382
2baed8786bd36a6657054d1cf6c43a
```

When initializing your database, you should store the passwords in this format. Use these values:

```
username =  sportslover
password =  paulpass93
```

```
database =
sha3_512$5d30ef0be96c48f98b1d31bc2fcabdcb$109d65b68002de0c534fa577e87d
48886eb14bb6fe19d927c43b3979460c9226db777cac4600abdfdc8ddc60e1e3e949e7
edcd4fd59cb79117c82274c079d8db
username =  traveler
password =  rebeccapass15
database =
sha3_512$926a7579a1f544f9bba336a226d963f5$c31254294edd666044eb83b0829d
1732f2a1da457a5b270248c99bc6f29efabfaf1235c3c91d1f4674494dde80ef44328d
d382e9849f53d916601195ccf6c66b
username =  spacejunkie
password =  bob1pass
database =
sha3_512$f029e972b0cb44baaa029e3f9c3ec2e0$a9b44929beb8f16ffffe8311326f
022bb0e5e490616c41bd880cb518f72aaca203dfd8fe5c652a05ab19fca811955b8787
c1fff099d2e92a59df9954a6e331a6
```

If a session already exists redirect the user to `/user/edit`. Otherwise, after adding the user to the database, redirect to `/login` and allow the new user to log in with their new credentials (do not log them in automatically after signup).

Edit Account page: `/user/edit` [sensitive]

The user should be able to change his/her firstname, lastname, password and email address (but not username). You should only change one field per request, and those fields should be the values in the `POST` request.

An example `POST` to `/user/edit` is:

```
"password1": "newpassword1",
"password2": "newpassword1"
```

OR

```
"firstname": "Mark"
```

Again, validate the input values on the server (details in Part 4).

User Login page: `/login` [public]

Here, a non-authenticated user can enter their username and password to login to a specific user account and become authenticated. A `POST` request to `/login` with `username` and `password` should sign in the user.

Upon successful login, redirect. If there is a url query parameter (for example: `/login?url=/the/prev/url`) the redirect to the URL. Otherwise, redirect to `/`.

Refer to Part 3 "Authentication" and Part 4 "Validation" for more details on successfully logging in users, and how to notify users when they incorrectly attempt a login.

My Albums page: `/albums` [sensitive]

This page is similar to its corresponding route in Project 1, in that it contains links to albums which the current User owns, as well as a link to the `/albums/edit` route. Note that, instead of using a URL parameter to input the username of the User's albums that we want to see, we are instead using the current session. You can navigate here by clicking on the 'My Albums' button at the index page.

Public Albums of All Users:`/albums?username=<username>` [public]

This page shows all of the public albums to a User, whether or not you are logged in. Here, you find links to view the albums, but no links to edit them. You can navigate here from the index page.

Albums page: `/albums/edit` [sensitive]

This is the `/albums/edit` page from Project 1. This page allows the user to add new albums, view existing albums, delete them or edit them. Remember that deleting an album should also involve deleting pictures in the album.

A couple notes:

- A user can only edit albums they created.
- There is no username parameter because we get that from the session cookies
- All new albums have private access by default.
- Deleting an album deletes user permissions for the album.

A user should only be able to view this page if they are logged in. Redirect the user to the `/login` page if they are not signed in.

Edit Album page: `/album/edit?id=<albumid>` [sensitive]

At the top of this page the user should be able to change the album permissions. There should be some way the user can edit a list of **other users** to whom he/she would like to give explicit access to view this album if it is private (Note: the permissions of the album's owner cannot be altered). If an Album becomes public, **all users that don't own the album but have permission to view it should lose said permissions.** You should also list the pictures in the album. Users should be able to delete pictures from the album as well as add new pictures.

Users should also be able to click on individual images and be directed to `/pic` from Project 1. Make sure to keep the `Album.lastupdated` field in the database updated whenever `Album.access` or `Photo.caption` is changed (as well as all album updates from Project 1). This implies that pic captions are editable (as described in `/pic`).

When granting a user permission, the form data should have the following:

```
op: "grant",
albumid: <albumid>,
username: "<username>"
```

When revoking a user's permission, the form data should have the following:

```
op: "revoke",
albumid: <albumid>,
username: "<username>"
```

When modifying an album's public/private access, the form data should have the following:

```
op: "access",
access: "public" OR "private",
albumid: <albumid>
```

View Album page: `/album?id=<albumid>` [sensitive/public]

This page displays the thumbnail view of an album's pictures just like the previous assignments. You can view this page if the album is public, you own the album, or the album is private and you have been given explicit access.

The album title should be at the top, along with the album's owner. The photos should be displayed in sequence order, each with its date, and a caption (if caption exists for that photo). Similar to project 1, clicking each photo should take you to `/pic?id=<picid>`.

View picture page: `/pic?id=<picid>` [sensitive/public]

This page displays a picture just like the previous assignment. It should have the caption, full-sized picture and links to previous and next picture.

You must be able to edit the caption. To edit the caption, make a form that `POST`s with the following information:

```
op: "caption",
picid: "<picid>",
caption: "<new_caption>"
```

If you do, make sure to update the `Album.lastupdated` field for the album that pic is in.

You can view this page if the picture is part of a public album, you own the album it is a part of, or it is in an album that is private for which you have been given explicit access. If the user does not have access to the album this picture is in, they should not be able to see the picture; an error `403` should be returned. You can only edit a picture's caption if it is part of an album that you own.

Logout page:`/logout` [sensitive]

This should destroy the session and redirect to the default home page. It should be a POST request.

Part 3: Authentication

In order to view a sensitive page, the user must be authenticated. On the `/login` page, you should use a form with username and password fields to send authentication information to the server for validation. Your server must handle the following error conditions:

- Username does not exist
- Password is incorrect for the specified username

If one of these login issues occurs, the `/login` page should be returned again, but this time with a descriptive error message at the top.

Part 4: Validation

All rules are specified in the format <description of error>, <exact text to return to users>

Your site must enforce the following rules. :

- The username must be unique (though case insensitive), "This username is taken"
- The username must be at least three characters long, "Usernames must be at least 3 characters long"
- The username can only have letters, digits and underscores, "Usernames may only contain letters, digits, and underscores"
- The password should be at least 8 characters long, ""Passwords must be at least 8 characters long"
- The password must contain at least one digit and at least one letter, "Passwords must contain at least one letter and one number"
- The password can only have letters, digits and underscores, "Passwords may only contain letters, digits, and underscores"
- The first and second password inputs don't match, "Passwords do not match"
- Email address should be syntactically valid, "Email address must be valid". Here is a regular expression which should check email validity well:

```
import re
if not re.match(r"[^@]+@[^@]+\.[^@]+", email):
```

```
                # handle an invalid email address
```

- Except for password, all fields (`username`, `firstname`, `lastname`, and `email`) have a max length of 20, "<field> must be no longer than 20 characters" (If `lastname` was too long, for example, the error would be “Lastname must be no longer than 20 characters”.

Appropriate validation errors must be found in your page in the following form:

```
<p class=”error”>
      **Error message goes here**
</p>
```

Where **Error message goes here** is replaced with the appropriate error message. For example:

```
<p class=”error”>
      Lastname must be no longer than 20 characters
</p>
```

You can assume that the user is acting in good faith: your goal is to prevent users from adding bad usernames/passwords, not to guard against motivated attackers who want to sneak a strange entry into your password database (which means you do not need to check things beyond above rules).

As mentioned in above sections, server-side validation of matching passwords (the two passwords the user entered when signing up/editing their information) is necessary. In the event of breaking any the validation rules, redirect to the page that the information was entered on with an error about User sign-up failing. The error to display is specified next to the description of the error.

Grading and Deliverables

Make sure that all these element IDs are present in your HTML templates:

- `/` (not logged in)
  - The link for login should have id `home_login`
  - The link for account creation (`/user`) should have id `home_user_create`
  - The links to public albums should have ids `album_<albumid>_link`
- `/user`
  - The input for username should have an id `new_username_input`
  - The input for firstname should have an id `new_firstname_input`
  - The input for lastname should have an id `new_lastname_input`
  - The input for email should have an id `new_email_input`

    - The input for your first password field should have an id `new_password1_input`
    - The input for your second password field should have an id `new_password2_input`
    - The submit button for updating should have an id `new_submit`
    - Follow the rules for validation input error as specified in the validation section; all error `<p>`'s need to have class `error` and the correct error message
- `/user/edit`
    - The input for firstname should have an id `update_firstname_input`
    - The input for lastname should have an id `update_lastname_input`
    - The input for email should have an id `update_email_input`
    - The input for your first password field should have an id `update_password1_input`
    - The input for your second password field should have an id `update_password2_input`
    - Follow the rules for validation input error as specified in the validation section; all error `<p>`'s need to have class `error` and the correct error message
- `/login`
    - The input for username should have an id `login_username_input`
    - The input for password should have an id `login_password_input`
    - The `<p>` for a nonexistent username should have id `error_username`
    - The `<p>` for a incorrect un/pw combo should have id `error_combo`
- `/albums`
    - Each link to `/album?id=<albumid>` should have id `album_<albumid>_link`
- `/albums/edit`
    - Each link to `/album/edit?id=<albumid>` should have id `album_edit_<albumid>_link`
- `/album`
    - Each link to `/pic?id=<picid>` should have id `pic_<picid>_link`
- `/album/edit`
    - The radio button for public album toggle should have id `album_edit_public_radio`
    - The radio button for private album toggle should have id `album_edit_private_radio`
    - The submit button for toggling album access should have id `album_edit_access_submit`
    - Each “remove access” submit button in the access table should have id `album_edit_revoke_<username>`
    - The text input for granting new access should have id `album_edit_grant_input`
    - The submit button for granting new access should have id `album_edit_grant_submit`
- `/pic`

    - The `<p>` that shows the caption should have id `pic_<picid>_caption`
    - The text input for editing the caption should have id `pic_caption_input`
    - The submit button for editing the caption should have id `pic_caption_submit`
- <all signed in pages>
    - The “Nav Home” link should have an id `nav_home`
    - The “Edit Account” link should have an id `nav_edit`
    - The “My Albums” link should have an id `nav_albums`
    - The “Logout” link should have an id `nav_logout`

NOTE: While some of the ids from project 1 were explicitly specified in above, you should maintain **all** of the ids we gave you in project 1 (except where noted as different); we can and will test you on them!

Code

Submit the following files to the autograder:

- `source.tar.gz` A tar archive containing your application source code.
- `tbl_create.sql`
- `load_data.sql`

How to create a tarball of your source code, from your git repository:

```
git archive --format tar.gz HEAD > source.tar.gz
```

We will post a link to the autograder when it is ready for this project.

In the `README.md` at the root of your repository please provide the following details:

- Group Name (if you have one)
- List the contribution for each team member:
  `User Name (uniqname): "agreed upon" contributions`
- Any need-to-know comments about your site design or implementation.

We will check the pa2 directory for your new secure photo album website. Based on PA1, your website should contain at least the following users with their albums. **Remember that in this project, we are hashing the passwords, so you will have to calculate the data for the initial input**.

- Username: sportslover, Password: paulpass93 - "I love football" (private), "I love sports" (public)
- Username: traveler, Password: rebeccapass15 - "Around The World"(public)
- Username: spacejunkie, Password: bob1pass - "Cool Space Shots" (private)

Your website may contain other users and albums, but please ensure that the above users and albums exist. Do not touch the files in pa2 sub-directory after the deadline.

As mentioned before, **Remember to commit your code into GitHub and the server, please do not modify your code after the due date - either on the repo or the server**, or else we will assume your submission is late.

Optional Exercises

The following exercises are completely optional, and **will not impact your grade**. The main reason for providing these additional exercises are to provide an opportunity to challenge yourself. Please take on the optional exercises after you have completed the rest of the assignment.

- When form is submitted in the New User page, send an email message to the new user confirming his/her membership and redirect them to the logged in home page. (HINT: check out flask-mail.)
- An additional class of "root-user". Anyone who is a root-user can edit anyone's album. There should be an administrative interface for giving/removing the root-user privilege. The `User` table should reflect this
- Use CSS to align images in rows and columns (no HTML tables allowed!) and make the website look more appealing.
- Implement 2-factor authentication