IEMS5718 - Internet Security and Web Development

Week 7 - Authentication, Authorization and Session

- Introduction to client-side storage
- Introduction to authentication, authorization and session
- GAE Tutorial: Using User Service, templates and static files

---

Warm Reminder

This chapter concludes our coverage of web development, which occupies the first-half of the course. Some of you may find materials of this week and Assignment 2 particularly hard, especially if you do not have solid understanding of materials from previous weeks.

Worry not though! If you are having trouble, please don't hesitate to go back to materials in previous chapters. Consider Assignment 2 a "**midterm**" for the course - you are expected to digest what you have learned so far to handle it!

1. Client-side Data Storage

Last week, we discussed server storage. This week, we will introduce client-side storage.

1.1 HTML5 Storage

- Before HTML5, storing data in web applications means sending the data to server. In HTML5, several new technologies are developed that allow apps to save data on the client

Pros	Cons
• Local: easy and fast access • Available offline	• More vulnerable • Can't access from multiple clients • Only for non-critical data

- 4 main types of Web Storage, IndexedDB, Web SQL Database and File Access
- IndexedDB, Web SQL Database are not well supported

1.2 Using Web Storage

- Web storage is simply a key-value mapping storage in the browser
- Two different types: local and session storage

Local Storage

- Per origin
- Limited size: 5MB (in Chrome)
- The data are sandboxed, meaning the data is only available to scripts loaded from pages from the same origin that previously stored the data and persists after the browser is closed
- No transactions, 'racing' conditions can happen
- Synchronous, be careful not to value to write is not too large

Add a value pair	`localStorage["name"] = value;`
	`# or`
	`localStorage.name = value;`
	`localStorage.setItem('name',value);`

Delete a pair	`delete localStorage['name']`
	`delete localStorage.name`
	`localStorage.removeItem('name')`

Serialize and deserialize data	`# use json, refer to Week 5`
	`# serialize an object and save to localstorage`
	`localstorage.tom = JSON.stringify({'name':'Tom'}) ;`
	
	`# deserialize`
	`var tom = JSON.parse(localstorage.tom);`
	`console.log(tom.name);`

* Definition of same origin: refer to Week 5 lecture notes

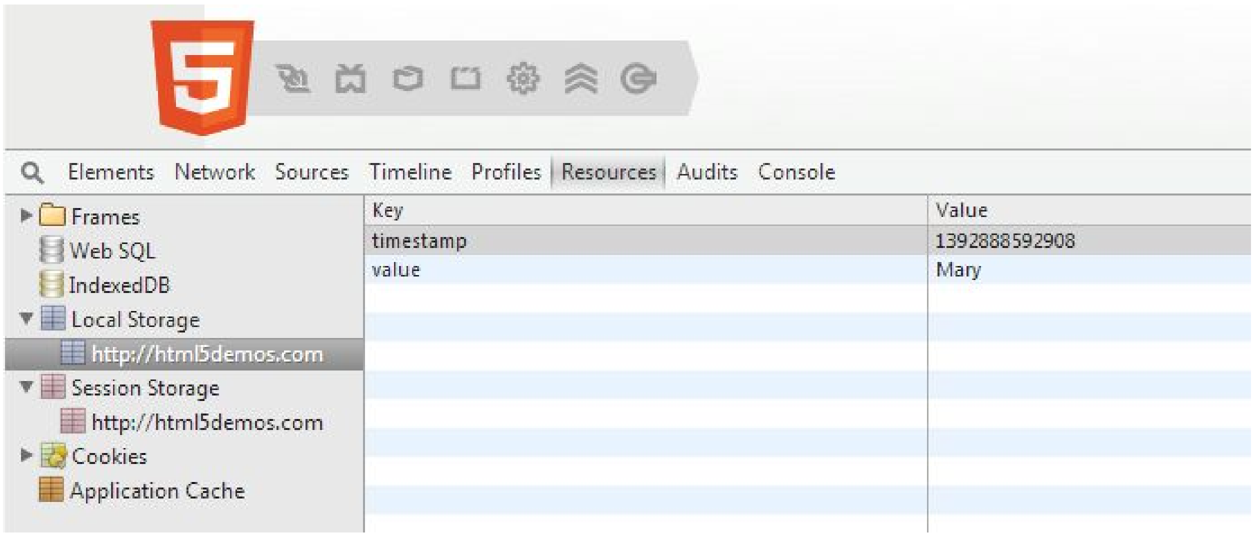
Session Storage

- The usage is similar to localStorage
- Per-page-per-window and is limited to the lifetime of the window.
- Allow separate instances of the same web application to run in different tabs/windows without interfering with each other

Find out the difference between localStorage and sessionStorage at this demo

View Web Storage in Developer Console (Chrome)

- Launch the developer console, go the Resources tab



Key	Value
timestamp	1392888592908
value	Mary

1.3 File Access

- The File APIs allow web applications to
  - Read files asynchronously
  - Create arbitrary Blobs (i.e. binary large objects)
  - Write files to a temporary location
  - Recursively read a file directory
  - Perform file drag and drop from the desktop to the browser
  - Upload binary data using XMLHttpRequest2
- Especially for handling binary data

Reference: Comparison of Web Storage

You may need to deal with binary data later in your project. We will cover this in later weeks.

2. Authentication, Authorization and Session

2.1 A Simple Explanation

Authentication -verifies who you are. There are many ways to verify who you are. In web applications, the most common way is using the combination of username & password

Authorization - verifies what you (as a user) is authorized to do. E.g. A Google user has access to his/her Google Profile

Sessions - Just as plain as its meaning, it means maintaining a dialog between the client (user) and the server

2.2 Sessions

Why we need sessions?

To jog your memory, web applications communicate with server and clients using the HTTP protocol which is stateless. A stateless protocol does not require the server to retain information or status about each user for the duration of multiple requests.

Therefore, we need methods to maintain a session (dialog), these are many session implementations, e.g. HTTP cookies, server side sessions

In order to understand how sessions are implemented, we need to understand what is HTTP cookies.

What is HTTP Cookies?

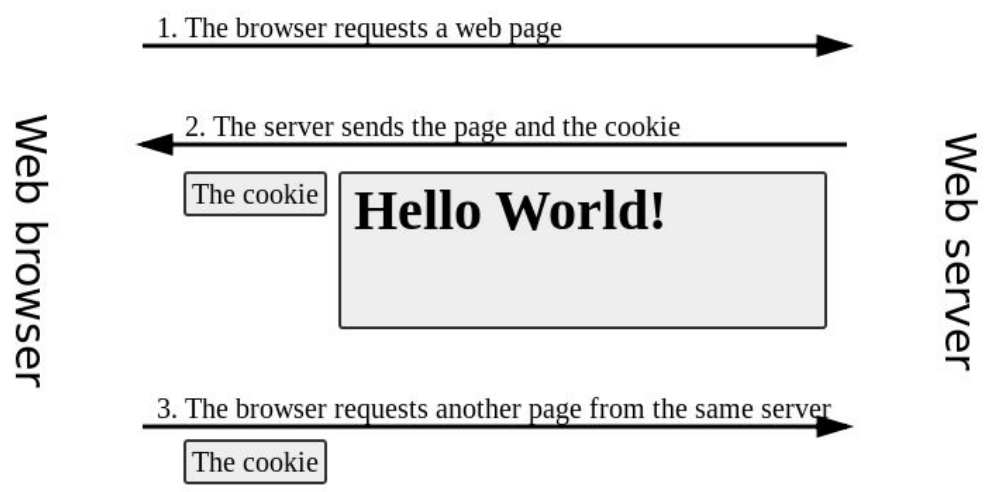
HTTP cookie is a small piece of data that a server requests a browser to store locally. **The cookie will be included in future HTTP request headers to the same server until it expires.**

- While cookies are stored in the browser, it is usually meant for use by the server, not the front end
- It occupies a limited size of at most 4KB and reside in the HTTP Header

The unique nature of cookie makes it ideal for maintaining session. Note that you won't store much in cookies - user has the right to **clear it anytime**.

Browsers usually support the following components:
- The name of the cookie

- The value of the cookie
- The expiry of the cookie
- The path the cookie is good for
- The domain the cookie is good for
- The need for a secure connection to use the cookie
- Whether or not the cookie can be accessed through other means than HTTP (i.e. JavaScript)


Web browser
1. The browser requests a web page
2. The server sends the page and the cookie
The cookie
Hello World!
3. The browser requests another page from the same server
The cookie
Web server

source

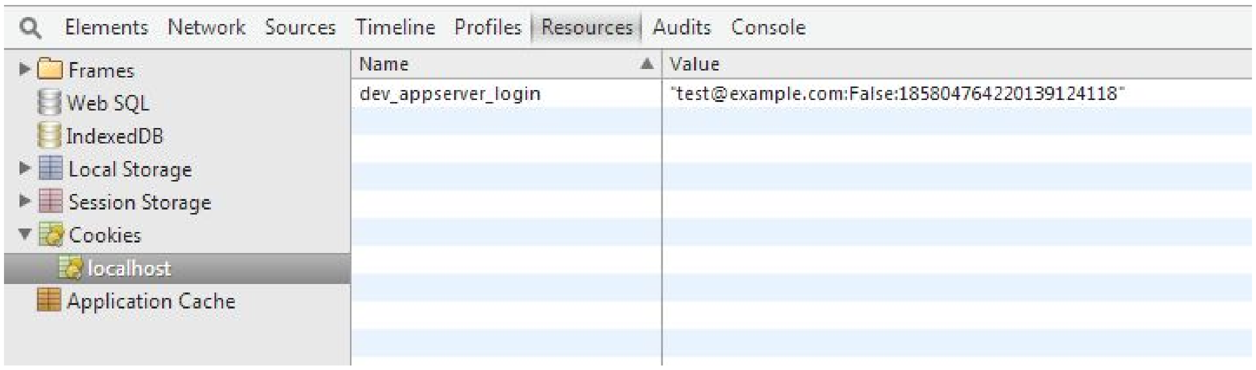
As one can see from the above diagram, the web server will send the client (browser) a certain cookie. The client (browser) will keep sending the same cookie in all subsequent requests to the server as an indicator. In layman terms it's like *"hey it's me again, do you remember this cookie you gave me?"*.

Common use

- Session cookies - maintains a session (e.g. a shopping session, a form-filling session - authenticated or unauthenticated)
- Authentication cookies - so when we use the same browser, we don't need to login every time (e.g. "remember me", remember user across sessions)
- Personalizations, e.g. remember users preferences like the locale of users
- Tracking cookies (persistent cookies), track long-term browsing behavior and history
- Third-party cookies, cookies sent from a different domain than the visiting sites, usually for serving online advertisements

View Cookies in Chrome

- Open Developer Console > Resources > Cookies



Name	Value
dev_appserver_login	"test@example.com:False:185804764220139124118"

Session & Cookies implementation using webapp2

- You are not required to implement your own session in your project
- If you are interested in using more advanced session in the future, check out the webapp2 framework

For example,

Setting a cookie on server	`# in webapp2, you can call`
	
	`self.response.set_cookie('key', 'value', max_age=360, path='/')`
	`#  e.g. here, max_age 360 means cookie will only last 360 seconds`

Setting a cookie in client using Javascript	`document.cookie="message=Hello";`
	
	`# You can also use jQuery cookie plugin`

Cookies are removed either when it has **expired** or **by the users using the browser settings**.

Security issues in Sessions and Cookies

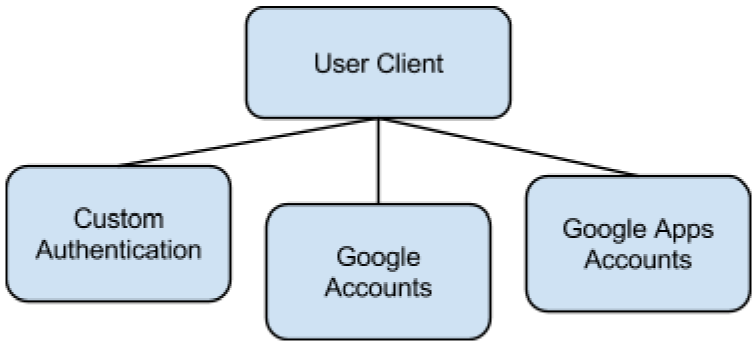
If sessions are not properly implemented in web applications, this will introduce a security issue called "**Broken Authentication and Session Management**" which is number 2 of the top 10 security risk in 2013 according to OWASP.org.

When a session is compromised, attackers can impersonate the users to do whatever the users could do legitimately. We will go into details of this kind of attack in lectures about security.

2.3 Authentication

Traditionally web applications all maintain their own authentication mechanism. Every site maintains their users account, username and password, in GAE we call this "custom login".

Custom login has to be implemented on your own, you can write it from scratch or use python modules written by other developers. Currently, there are two built-in authentication methods on GAE.



Using Google Accounts & Google Apps Account

Google Accounts is Google's unified sign-in system. All a user needs is a valid email address (it doesn't need to be a Gmail address!) to sign up for a Google Account. This also includes Google Apps accounts which have transitioned to the new Google Apps infrastructure.

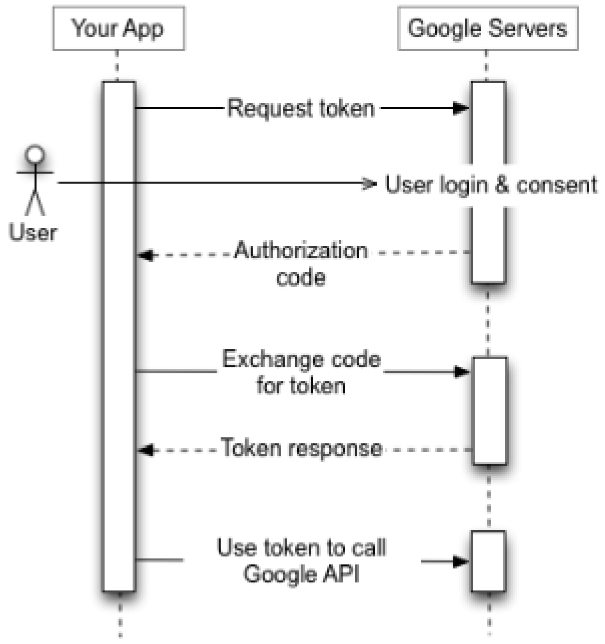
Users of **Google Apps** can choose to restrict all or part of their web application to only those people who have a Google Apps account managed by their domain.

2.4 Authorization

Authorization is about permissions, e.g. File permission on linux. In a web application, a user is usually authorized to view his/her data, the admin user is authorized to access data across different users. It is considered to be a serious security issue if users are able to access or modify data without authorization.

OAuth 2.0

The web contains lots of data in different web sites. To share these data, a authentication and authorization standard is necessary, this standard is OAuth 2.0. For example, Google providers many Google Data APIs that allow other applications to access/modify their data with OAuth 2.0. For example, a 3rd party web application can access Google+ data with OAuth 2.0.



- An application requests to use a user's Google data
- The user is redirected to a Google URL that request login and consent
- The Authorization token is return from Google
- The application will store this token and use it to request data from Google APIs directly
- Application will need to refresh token when necessary or request again if the token expires

Reference:
- https://developers.google.com/identity/protocols/OAuth2
- Google's OAuth 2.0 Playground

3. GAE Tutorial

In this tutorial, we will learn how to use

- User Service
- Using templates and static files

3.1 Using User API

Authenticate users with the User API in Python

* Be careful with indentation when you copy and paste code directly

Import the User API	`from google.appengine.api import users`

Create a handler for the login page	`class MainHandler(webapp2.RequestHandler):`
	`    def get(self):`
	`    # define the method here`

Get the user object	`# show the landing page`
	`user = users.get_current_user()`

If user object exists	`if user:`
	`     # login is found, redirect to the form`
	`     self.redirect('/edit/form')`

If no user object found, render the landing page and show link to login	`else:`
	`    greeting = ('<a href="%s">Sign in or register</a>.' %`
	`users.create_login_url('/'))`
	`    self.response.out.write('<html><body><div>Welcome to Form`
	`Demo</div>%s</body></html>' % greeting)`

Add logout link in appropriate page	`# add the logout link in your application, the url '/edit/form/ in`
	`this case`
	`# this will return a logout link, and redirect users to '/' after`
	`logout is successful`
	
	`users.create_logout_url('/')`

3 important methods	`users.get_current_user()`
	`users.create_login_url(dest_url)`
	`users.create_logout_url(dest_url)`

Please check out: Users API

Exercise 1

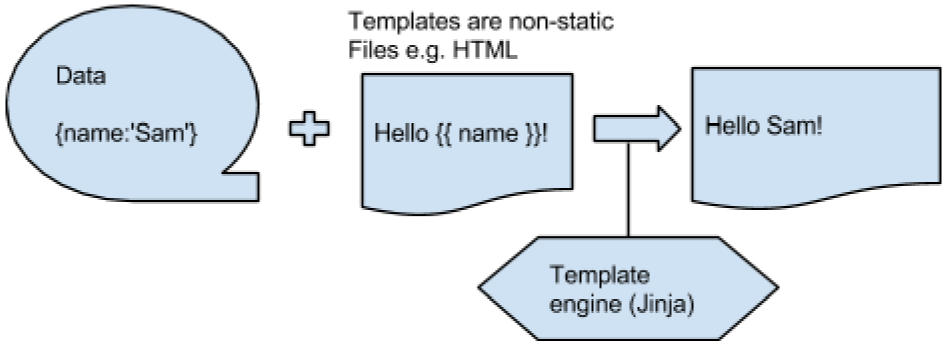
- Go to Google Cloud Playground, select "**Users API**" from the list of projects, click "Copy" and open it (make sure you have signed with a Google account)
- Run this demo in a **new window**, you will be asked to allow access from this app
    - *If you just press Run nothing will happen*
- Refer to the Users API , try to display your user id, email, create logout link etc.
- Take a look at the cookies in the Chrome Developer Console (You won't be able to understand what Google is doing exactly with the cookies, but at least you should know where to check them out).

3.2 Using Templates and Static Files

Web application is all about rendering HTML. When rendering complex HTML, embedding HTML code in python is messy and hard to maintain, also causes clutter in the business logic with HTML code.

Therefore, we keep the HTML in a separate file and use special syntax to insert data into it. The file is called template and the python API that renders templates is called Jinja2 templating engine.

* Django is also another templating engine that is included in GAE. In this course, we will use Jinja2



Import the modules in app.yaml	`#app.yaml`
	
	`libraries:`
	`- name: webapp2`
	`  version: latest`
	`- name: jinja2`
	`  version: latest`

Import the module in your python app	`# main.py`
	
	`import jinja2`

```
import webapp2

JINJA_ENVIRONMENT = jinja2.Environment(
    loader=jinja2.FileSystemLoader(os.path.dirname(__file__)),
    extensions=['jinja2.ext.autoescape'],
    autoescape=True)
```

Define what values to pass to the template as a dictionary. Here `template_values` is set up and will be used when we render the template.

```
# In main.py
# Example 1
template_values = {'nickname': 'Sam','age' : 44}

# Example 2 - ok to pass objects too
user = get_current_user()
template_values['user']= user

# Example 3 - cat is also an object (a datastore entity)
cat = Cat(name='Coffee', age=5)
template_values['cat']= cat

# Example 4 - ok to pass a whole list too
greetings = greetings_query.fetch(10)
template_values['greetings']= greetings
```

Render the template; note how template values are passed to `template.render`.

```
template = JINJA_ENVIRONMENT.get_template('templates/form2.html')
self.response.write(template.render(template_values))
```

```
Access values passed to the template
```

Variables
- Use **{{ }}**

```
# display  string or integer
Hello {{ nickname }}! Your age is {{ age }}

# display object
Hello {{ user.nickname() }}!

#  display  entity
Hello, your cat is {{ cat.name }}!
```

Expressions
- Use **{% %}**
- In Python-like syntax but without indentation

```
# if else statement
{% if age >= 18 %}
  Some 18+ only content here...
{% endif %}

# display query results (from example 4 above)
# use for loop
 {% for greeting in greetings %}
      {% if greeting.author %}
        <b>{{ greeting.author.nickname() }}</b> wrote:
      {% else %}
       An anonymous person wrote:
      {% endif %}
      <blockquote>{{ greeting.content }}</blockquote>
 {% endfor %}
```

Reference on templates

Exercise 2

Go to the template demo here. Try the following changes:

- Add back **application: your-app-id** to the top of app.yaml
- Instead of "Login", change it to "Login Here"
- Instead of having the login name at the very bottom, put the login link at the very top.
- Add a paragraph that says "You are recommended to login before you post in the guestbook" under the text area if the visitor hasn't logged in.
- Note that in the demo there is a "`{{ url|safe }}`". Here **safe** is a filter which tells the server that this url need not be escaped (being "HTML-safe", also note the tag "`{% autoescape true %}`"). Try sign the guestbook with "<b>I am bold</b>" and see what happens. How would you change so that HTML input can be displayed? What problem will it cause?

Tags and filters are more advanced topics, and will not be covered in our basic introductions. Please consult template documentation for more information.

Using static files

HTML templates is non-static because the content changed dynamically, static files are files that remained unchanged in the application, e.g. css files, javascript files and images

However, remember GAE does not reference files like a simple web directory. In order to serve these files, we have configured URL mappings in the app.yaml

Map by directory: static_dir	
	```
	# in app.yaml

	handlers:
	# All URLs beginning with /css are treated as paths to static files # in the css/ directory.

	- url: /css
	  static_dir: css

	- url: /js
	  static_dir: /js
	```

Map by files: static_files	
	```
	handlers:
	# All URLs ending in .gif .png or .jpg are treated as paths to static files in the static/ directory. The URL pattern is a regexp, with a grouping that is

	# inserted into the path to the file.
	```

```
- url: /(.*\.(gif|png|jpg))$
  static_files: static/\1
  upload: static/.*\.(gif|png|jpg)$

# /a.png will map to static/a.png
```

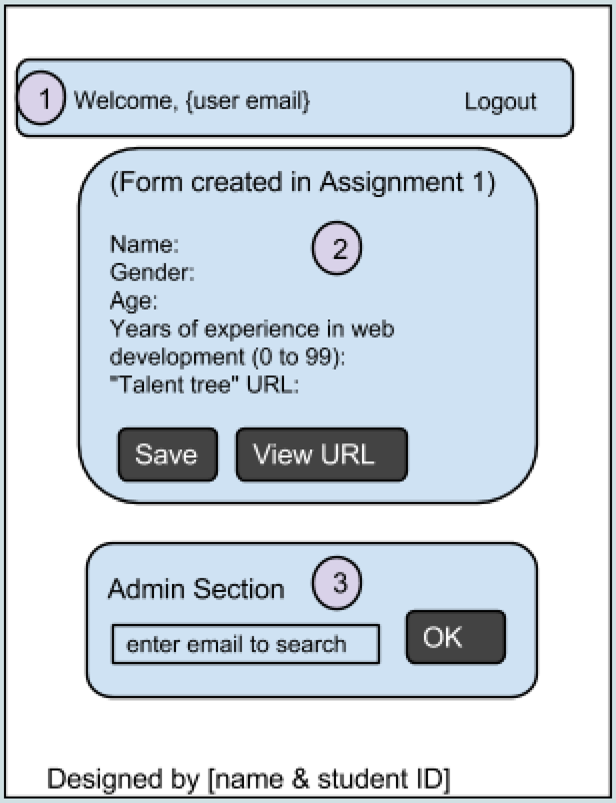
* Basic knowledge on regular expression is required in order to create more complex URL mappings

Reference: Static Files

Assignment 2

Using Ajax, Users Services and Templates

Extend your Assignment 1 to really connect to a Google App Engine server to save the data. Here is a framework (link) to start your assignment. The requirements are as follows:



Start with your form from Assignment 1. The form should now be modified to follow the structure on left:

1. Header - include a message welcoming the current user (user id = user email) and a logout link

2. Content - The form
    a. Use AJAX to submit form data to server
    b. When user clicks "Save", the confirmation is shown when server has really saved the data

3. Advanced Feature (Bonus):
    a. Create an admin section only visible for admin users.
    b. Click the "OK" button to search data by email (Use AJAX).
    c. If an entry is found, return the data in JSON format, then use javascript to fill in the form.
    d. Admin can modify the data and click "Save",
    e. If no entry is found, a message will tell the admin that no entry is found for this email.

Additional authentication requirements

- You should use the login page from the demo; user must first login to view the form.
- The form should be filled with previously saved data (using template, as introduced this week) from the same user if available, submit again will overwrite the previous form
- The logout link will take the user back to "/", the handler for "/" is already in the demo

About the displaying user's email

- When using the User API, user is required to login with an email
- This email should be used as the **Key Name** for the Form entity, therefore
    - Your datastore model class ("Form" in demo) do not need to have an email field
    - The HTML <form> does not need to contain an email input

You are recommended to use GAE SDK for easier development and testing.

Submission

Zip your application folder and name as **"<student id>-A2.zip"** and submit to elearn

Submission Deadline: **13 March 2016**

Hints for Assignment 2

- You can either add stuffs to your original form from the demo html or copy part of your form to the demo html to start. Either way is ok.
- The advanced feature (the admin function) gives **very little bonus** (2 mark) but **requires considerable work** (> 100% more work). Please consider doing it only if you believe you are really good in server programming.
- You will not get bonus for changing or modifying the login page.
- Again, you might need to brush up your HTML when you do the assignment. **USE THE DEVELOPER CONSOLE TO DEBUG.**
- Make sure your form is working before you test any server side programming logic

Last Updated by Dr Hui Ka Yu & Sandy Lam, 2016