**This is the ‘home page’ or landing site for the ktesa project. It is purely html and serves as a starting point for visitors, providing choices for how to proceed. It also allows for registering users on the site, further allowing users to create and edit new pages.**

* There are two forms controlling where the user is directed when submitting:

1. For standard viewing of pages, the <form id=”mainfrm”> invokes the script, pages/mapPg.php. That script will direct the user to either a ‘Table Only’ page, a ‘Map + Table’ page, or a ‘Full Page Map’ page. Geolocation options are also determined by the user here *(ref. Design Description: mapPg.php)*.
2. For user registration and optional features, the <form id=”auxfrm”> is utilized, but the action=”#” keeps the user on the page and allows javascript to determine additonal page features:

* The interactive features of the page are controlled by javascript. The javascript for this page is held in the file: scripts/main.js; it interacts with the html:

1. Clickable text providing more details on the operation of the site, <p id=”turnon”>, displays the <div id=”more”> when clicked; text at the end of that div, the element <p id=”turnoff”>, will hide it again.
2. At the end of the page, a <div id=”closer”>, when clicked will bring up an email to the site master for the user to comment or explain any difficulties encountered.
3. User registration:
   1. Contained within the <div id=”logins”>, there are two boxes provided for user registration (if the user has not already registered). This div is hidden once successful login occurs:
      1. User name: In the <input id=”usrid” />, a registered user can enter the registered user name. Note: after registering, if cookies are enabled on the browser, this div will not display, and instead a message welcoming the user back will be shown (text formed by javascript using user’s login name). No further logins will be required in that case, and the ‘Sign me up!” link will not be displayed.
      2. Password: The password established by the user during registration is to be entered here. A strength indicator for the password is provided during registration; <input id=”upass” />
   2. If a user has not yet registered, he/she can click on the link “Sign me up!” (refer to 3.e.ii).
   3. Site master can simply enter the site master password in the password box without supplying a name.
   4. Once signed in, the behavior of the page is as described later.
   5. Contained within the <p id=”loggedin”> element are the
      1. ‘Display Options’ button, <input id=”sub2” /> which submits the form (to ‘self’ via javascript), and
      2. ‘Sign me up!’ link, which directs the page to admin/Registration.html.
4. ‘Back door entry’ for site master if user is having a problem and needs the site master to verify the problem and troubleshoot with the user’s id. The <p id=”mover”> element contains the text “[ Site Master Entry ]”, and the input box <input id=”mstrpass” /> in which to type the master password.

*Registration/Authorization/Cookies:*

* The javascript is loaded after the page (DOM) has been loaded. It initializes the variable ‘usr\_type’ to ‘unregistered’. The flag enabling display of the ‘backdoor’ is initialized to ‘false’. Then it looks for the existence of cookies.

1. First, the code attempts to load the ‘master’ cookie (nmh\_mstr). If it is not empty ( “ “ ), usr\_type is set to ‘mstr’. Note that the value of the cookie is not used, only the existence of it. Once set to ‘mstr’, the <div id=”logins”> is hidden as logins are no longer required, and the <p id=”loggedin”> section is displayed with the “Sign me up!” link turned off (via the <span id=”reg”> element). The ‘Back Door’ <p id=”mover”> element is hidden.
2. The code next attempts to load the ‘user’ cookie (nmh\_id). If present, the usr\_type is set to the value of the retrieved cookie (dependent on user). A message is then added to the page stating “Welcome back (user); you are now logged in…”. The user-available buttons & functions are now enabled and displayed on the page (function ‘usr\_login\_display()’). That function:
   1. Hides the <div id=”logins”>;
   2. Hides the “Sign me up!” link (via the <span id=”reg”>);
   3. Displays the ‘Back Door’ input and sets the ‘backdoor’ flag to true.
3. The page remains static at this point until the user takes further action.

* When there are no cookies, or the browser has cookies disabled (usr\_type remains as ‘unregistered’), a user can either (1) sign up (if not yet registered), or (2) can login using the provided input boxes (<div id=”logins”>).

1. When a new user clicks ‘Sign me up!’, the new page admin/Registration.html is displayed, where the user can enter the requested data. The data is submitted via a form whose target is ‘create\_user.php’ (in admin). User interactions are processed by the script user\_val.js (in admin) and by the jquery scripts: scripts/jquery.validate.min.js and scripts/jquery.validate.password.js, which were downloaded to perform the validation tasks. The validate object: .validate({ rules {}, messages {}}); is attached to the form submit with the form id=”registration” in the script included in the <head> section of the html. Check online documentation for the operation of this code (user\_val.js is explained below).
2. A section designated ‘Required Information’ contains the following html:
   * 1. First name <input name=”firstname” />
     2. Last name <input name=”lastname” />
     3. User-name <input name=”usr” />
     4. Password <input name=”password” />
     5. A div where the
        1. Password message and strength bar are displayed (message is ‘Weak/Good/Strong’ for the password)
        2. A ‘Confirm’ div with confirm <input name=”confirm\_password” />
     6. Email address <input id=”umail” name=”email” />
3. A section for ‘Optional’ Information, including:
   * 1. Facebook url <input name=”facebook” />
     2. Twitter handle <input name=”twitter” />
     3. A text area for entering any miscellaneous personal info <textarea name=”bio”>
4. A button to “Submit My Info” <input id=”setuser” />
5. A button to ‘Clear All Fields’ <input type=”reset” />

The javascript performs the password strength checking and email validation, which is set up by the function “validateEmail”. The function is called when the <input id=”umail” /> is changed by the user.

When the user selects the “Submit My Info” button, the form submits and the user is directed to a new page, ‘create\_user.php’. Here the user is advised of successful registration (unless errors are detected) and is provided a link back to the main page (“Main Page Link”), where he/she will be logged in as the new user, and the “Display Options” button will be available to perform user tasks. The javascript file ‘cookie\_check.js’ (in admin) tests the browser to see if cookies are enabled. If not, a message is displayed to the user that “Cookies are not enabled on your machine, therefore, you will also be required to enter your user password”. If enabled, a 365-day expiration date is set for the cookie. This script saves the user-entered data in the mySql table ‘USERS’.

1. When logging in as a registered user, the user enters the registered username and password to activate the “Display Options” button. User validation begins when the form is submitted (user hits the return button after entering name/password). This triggers the jQuery .submit() function for <form id=”auxfrm”>. The function first tests for the master password code.
2. If the master code has been entered, or the master cookie is set, or the ‘backdoor’ value matches the master code: the page settings are adjusted as follows (Note: no name is required when master password is entered in the username/password boxes):
3. The user button section is turned off; (useful when entering as a ‘backdoor’);
4. The master button section is turned on (all buttons visible);
5. The <div id=”logins”> is turned off;
6. The “Sign me up!” link (id=”reg”) is turned off;
7. The <p id=”loggedin”> element (‘Display Options’ button) is turned off;
8. The ‘back door’ box is turned off.
9. All buttons for editing/creation/hiding are enabled.
10. Otherwise (it is not a master), the function checks the browser to see if cookies are enabled. (validateUser() function is detailed later)

* *If cookies are not enabled*:
  + 1. The username/password boxes are checked to be sure one or both are not empty, and issues appropriate warnings if so.
    2. The function ‘validateUser()’ is called with the argument for ‘setcookie’ set to false. If the function returns with the global variable ‘valstat’ true, the global ‘username’ is set to the user’s login name; ‘username’ is used to pass information to the pages called when an option button is selected.
* *If cookies are enabled*, the script checks to see if one already exists for the user:
  + 1. If a cookie does not already exist, the validateUser() function is called with the ‘setcookie’ argument set to true, so that on future visits to the page the user will already be logged in (since cookies are enabled). As in the above case, if the function returns with the global variable ‘valstat’ true, the global ‘username’ is set to the user’s login name. The password box is cleared and the user option login info is turned off.
    2. If a cookie already exists, the global ‘username’ is already set to the login name (the cookie value). In this case, the user display options are turned on via the function ‘display\_usr\_opts()’.
* Operation of the validateUser() function:

1. validateUser() accepts the username, user password, and setcookie flag as arguments.
2. Javascript (jQuery) ajax is used to access the php script, admin/authenticate.php. The data passed to that script is in the form of an object whose keys are ‘nmhid’ (with a value = the username argument) and ‘nmpass’ (with a value = user password argument).
3. Simultaneous with the ajax request, an interval timer is set to check to see when the ajax request is finished processing. When done, the interval timer is cleared, the ajaxDone flag is returned to false, and, if authentication was successful (ie ‘usr\_type’ is ‘qualified’), the ‘welcome back’ message is displayed with the user’s username, the options are enabled, and, if the ‘setcookie’ flag is true, a new cookie is set.
4. The admin/authenticate.php script: a simple php script that accesses the USERS table and examines it to see if the username matches any of the entries. If there is no match or multiple matching usernames are extracted, the script returns ‘FAIL’ to the calling ajax routine. If there is exactly one match with the supplied username, the password is verified. The USERS password is encoded and the php ‘password\_verify’ function is utilized to compare with the supplied password. If it matches, the routine returns ‘LOCATED’ to the calling ajax routine, otherwise (doesn’t match) it returns ‘BADPASSWD’.
5. When the authenticate.php script completes, the ajax routine processes the ‘srchResults’, or output described above. If the expected ‘LOCATED’ string is returned, usr\_type is set to ‘qualified’. If ‘BADPASSWD’ is returned, the password box is cleared and a message is posted to the user that there was not a match with the registered password. If ‘FAIL’ was returned, the message is posted that the user info could not be located. In all of these cases, the next event is to set the ‘ajaxDone’ flag to ‘true’, so that the timer (see 3. above) can be cleared.