Application Use Cases:

1. **View Public Info**: All users, whether logged in or not, can
   - view list of topics
   - select a topic and see list of clubs (cname, desc) that are about that topic
   - see list of public events occurring in the coming days.
2. **Login**: User enters pid, x and password y via forms on login page. This data is sent as POST parameters to the login-authentication component, which checks whether there is a tuple in the Person table with `pid`$=x$ and the `passwd` $= md5(y)$
   - If so, login is successful. A session is initiated with pid stored as a session variables. Optionally, you can store other session variables. Control is redirected to a component that displays the user's home page.

   - If not, login is unsuccessful. A message is displayed indicating this to the user.
3. **Display Home Page:** Once a user has logged in, ClubHub will display her home page. Also, after other actions or sequences of related actions, are executed, control will return to component that displays the home page. The home page should display
   - Error message if the previous action was not successful,
   - Some mechanism for the user to choose the use case she wants to execute. You may choose to provide links to other URLS that will present the interfaces for other use cases, or you may include those interfaces directly on the home page.
   - Any other information you'd like to include. For example, you might want to include clubs the user belongs to, events she's signed up for, etc on the home page, or you may prefer to just show them when she does some of the following use cases.

   After logging in successfully a user may do any of the following use cases:
4. **View My Events**: Provide various ways for the user to see upcoming events of interest. These can be included as a single use case or as separate use cases. The default should be showing events for the current day and the next three days. Optionally you may include a way for the user to specify a range of dates. In each case, the display should include the e_name, date, time, location, and name of sponsoring club.
   - upcoming events for which this user has signed up
   - upcoming events that are public or are sponsored by clubs to which this user belongs
5. **Sign up for an event:** User chooses an event that they are eligible for (because they belong to the sponsoring club or because it's public) and signs up for the event. You may find it easier to implement this along with the **view event** use case. If the user has previously signed up for the event no additional action is needed (but ClubHub should not crash).
6. **Post a new event:** User enters ename, date, time, description, location, sponsoring club, and indication of whether event is public. ClubHub records the event with a new `e_id` and notes the sponsoring club. **The user must be a member of the club with role "admin" to post an event!** If not, ClubHub displays a meaningful error message.
7. **Check club's events :** For each club for which the user has "admin" role, ClubHub displays upcoming events (eid, ename, sponsoring clubs) and the number of people who have have signed up for each. (For a little **extra credit**, use an outer join to include events for which no one has signed up.)

8. **Logout:** The session is destroyed and a "goodbye" page or the login page is displayed.

9. Specify and implement some additional ClubHub features. If you are working alone, you only need to do one more feature. If your team has n members, you should do $2n - 1$ additional features (two additional features per additional team member). You must propose these features in your progress report (see below.) They must be interesting features that are specifically related to ClubHub (such as posting and viewing comments), not generic features, like a registration page. You should design test data that will allow you to demonstrate that your implementation of these features works, including unusaul "corner cases".

Additional Requirements:

- You should implement ClubHub as a web-based application. If you want to use a DBMS other than MySQL, SQLserver, or Oracle or to use a programming language other than PHP, Java, or C#, please check with me before March 30. You will need to bring the host computer to the demo/test session at the end of the semester or make the application available remotely over the web.
- **Enforcing complex constraints:** Your ClubHub implementation should prevent users from doing actions they are not allowed to do. For example, ClubHub should prevent users from signing up private event sponsored by clubs to which they do not belong. This should be done by querying the database to check that the user belongs to the club, before signing him up for the private event. You may *also* use the interface to help the user to avoid violating the constraints. For example, you could provide a list of events for which the user is eligible and allow them to select one.
- When a user logs in, a session should be initiated; relevant session variables should be stored . When the member logs out, the session should be terminated.
- Each component executed after the login component should authenticate the session and retrieve the user's pid from a stored session variable.
- *You must use prepared statements if your programming language supports them.* If your programming language does not support prepared statements, Free form inputs (i.e., text entered through text boxes) that is incorporated into SQL statements should be validated or cleaned to prevent SQL injection
- You should take measures to prevent cross-site scripting vulnerabilities, such as passing any text that comes from users through `htmlspecialchars` or some such function, before incorporating it into the html that ClubHub produces.
- The user interface should be readable, but it does not need to be fancy.
- For help in debugging and evaluating your project, you may want to echo each query that the application executes to a file or to the html that the ClubHub generates.