1. Interfacing your product with LinkedIn/Facebook.

To allow users to log in with their Facebook account, UMass Bookshare would have to go through a few steps. First we would have to make the actual 'Log in with Facebook' button on our app. The code for this is available on Facebook's developer site. We would have to also integrate authentication which is also part of the developers site. If we use Facebook's authorize() function we would not have to encrypt the passwords through our web app because it would just be authenticating and logging in from Facebook. The difficult part of interfacing with Facebook is actually retrieving the information from the Facebook profile to our UMass Bookshare profile. To do so, I would imagine we would have to retrieve Facebook's profile data in the form of a array (this is what I'm guessing it would be because that's how UMass Bookshare does it) and pulling in the fields to rendering it onto the profile page. To render it correctly we would need to match up the fields with our database fields. Facebook might call a user's institution, school in their database while we call it institution so we would need to equalize it in our backend. We would need to use Facebook's access tokens which allow us to use the data from the Facebook users. Once we figure out how to do that then that's all that's needed to interface Facebook with UMass-Bookshare. The integration and understanding of Facebook's API would be the tough task, once we know how that works, pulling data from it would just require more research.

*Ted said we could use Facebook as an example because it made more sense

2. **Scaling it to handling 300 million users**

Scaling UMass-Bookshare to handle at least 300 million users would present many issues. First and foremost the web app itself would not be able to retain it's visually appealing properties. The reason why is because if there are millions of users interacting on the web app, whether it be buying, selling, etc. then naturally UMass-Bookshare has to process their demands. For example, if a user inputted a search query and it returned 10,000 results, UMass-Bookshare would place all of that on one single page. That isn't practical and realistic because even if that somehow did work it would require to filter out which listing they want. To fix that we could do a variety of things. The simplest solution would be to separate the results into different pages after a certain cut off. Another solution would be to allow the user to make their queries more precise by adding filter options such as newest listing, containing exact words, etc. Both should work for the sake of being visually appealing and helping it be more user friendly.

Scaling to handle 300 million users would also require UMass-Bookshare to have more servers dedicated to the web app. If we even attempted to handle 300 million users on our current server(s) on AWS it would just crash and burn. We would need to add more web servers, load balance correctly, and even after that we would most likely still have to recode some parts of the app to make sure it can handle everything so it looks like everything is running concurrently.

Security becomes a huge concern as well when scaling up with a web app. As of right now UMass-Bookshare simply uses a node module called bcrypt to encrypt passwords. When there are millions of users we would most likely need a dedicated server(s) to just encrypt and authenticate users. An issue with using bcrypt with large scale web apps is that the response time could/will be slower. We would also have to make sure we have enough money to run bcrypt at an acceptable speed.

Lastly, we could also split the sharing load by regions. It would make more sense for it this way because UMass Bookshare connects people locally. It would not make sense for student from California meet up with students from UMass.

3. **Identifying and removing or blocking bot and spam users**

There are many ways to deal with bots and spam users. The first case is that we could put a limit to how often a user can post a listing by putting a timer in app.js that checks if the user has recently posted a listing. The problem with that is that it becomes harmful for regular users because now they have to wait to post say five listings. An alternative is to check if a bot or spam user rapidly filling out listings and creating them. If they are able to create a listing under inhumane times we could simply send a notification through email telling them they have been banned for X amount of time, this would ultimately require a timer as well. The best solution in my opinion would be a validation at the end of any create functionality add a CAPTCHA at the end which would help tremendously with deterring bots and spam users. By adding a CAPTCHA that would prevent the need to identify a spam user or bot because they would have to verify that they are indeed human and even if they got past the account creation they would need to verify at the end of every action so it wouldn't be practical for them.