Group Project Developer Review

Sylvia Unemna, Daniel Harris, Gwang-Seop Shin, Caleb Graves

Written by: Caleb Graves and Daniel Harris

The project presented us with a significant number of challenges and changing ideas about how to implement things as the development stages progressed. There were many steps where the easiest way to do things was not the cleanest or the most efficient, but with the considerations of restrictive time constraints and limited package knowledge I opted for the easy way. I will highlight a couple of the major issues I found and how each of these issues could be resolved for the project to advance.

Major Issue 1: SQLAlchemy and Insufficient Interfacing

The primary issue here is that before I even realized it was happening, SQLAlchemy had infected our entire codebase. At first I planned on using SQLAlchemy for our main classes and having all the methods be totally separate, but a quick glance through any method shows a full integration with SQLAlchemy. There are a couple of big problems here. The first is that if we were asked to use something other than SQLAlchemy – MongoDB for example – we'd have to code the entire project again from scratch because there's simply no way of extricating SQLAlchemy from our code. The second is that we don't have visibility or understanding of SQLAlchemy sufficient enough to trust that what we're doing is correct, scalable, and efficient.

For advancing to the next stage of the prototype, I'd interface off SQLAlchemy from the rest of the project entirely. I'd have a primary module containing our Post, User, Subscription, and Group classes and all our methods, but there would be a separate "database relation" module that would contain all the SQLAlchemy ORM classes. Methods and classes in the primary module would simply operate on the primary module's data, and the interface would allow us to link from the primary module to the table data through the SQLAlchemy module. This would allow us to replace the SQLAlchemy ORM with any other database system we prefer as long as we maintain the proper interface properties.

This is currently the most significant problem with our project.

Major Issue 2: Error Handling

Right now our code relies heavily on jinja to be able to make sure we're getting the data we want where we want it. There's not a proper interface between what jinja delivers to us and how we process that data, so there's a lot of funky hacks to ensure we get what we need. For example, to ensure a post isn't created with blank entries, if jinja gives us a POST request with a blank entry our code will simply flash a message that all fields need to be filled and keep the user on the same webpage. The actual __init__() class methods or any of the main methods don't do any kind of actual error checking, they just assume that what jinja is giving us is right if it hasn't raised any flags. This came up during testing where I couldn't go to the Register User page and register a user with no username or password but I could simply call the new user __init__() method with blank data and the code would happily generate a bad user.

To fix this, I'd maintain some separation and a cleaner interface between jinja and flask. For example, if a user tries to register with no password: instead of having flask check if jinja has a proper POST request with all entries filled, I'd take the POST request, generate a new user and have automatic error handling in our user class to ensure that if it's handed incorrect data it will reject it and return an error that can be passed back to the user. Moving the checking from the flask/jinja interface into the main python classes themselves will result in easier to understand, more stable, and easier testable code.

Major Issue 3: Incomplete Group Operation

We had envisioned groups to work like 'threads' on similar forum style web pages. In the prototype users should have encountered a list of group discussions on the home page which they may join to view posts within. Due to extensive time used trying to resolve errors in the source code, the idea of how to handle groups became ambiguous and because of time restraints was ultimately omitted from the homepage. In its place a stub for 'Group Discussions' was left. This means that all posts are now visible on the home page, and cannot be restricted to a specific group as intended. In this version of the prototype groups may be created and joined, however no visible tracking of such events exists.

In future versions of the prototype the option to create posts on the homepage should be replaced with a list of groups. There should be a default group called 'General' which every new user will already be a member of. Users should be able to create their own groups and join existing groups which only members can view and post in. Created groups should be visible on the homepage list so users may interact with them accordingly.