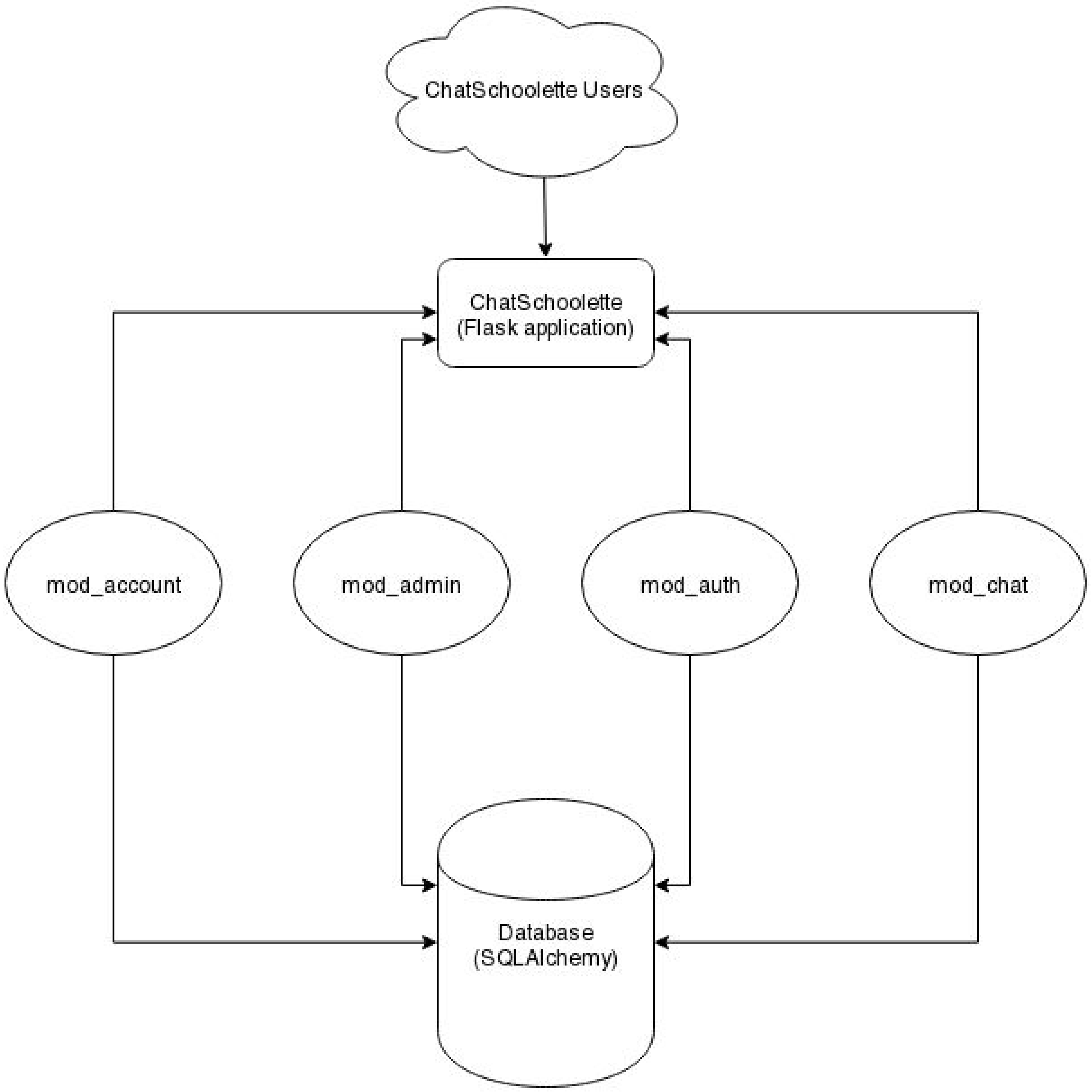
ChatSchoolette

Sprint 2 Incremental and Regression Testing

Team 1: Logan Gore, Kyle Rodd, Sang Rhee, GeonHee Lee, Stephen Hong, Tyler Springer

Classification of Components



ChatSchoolette is comprised of four major modules: ACCOUNT, ADMINISTRATION, AUTHORIZATION, and CHAT.

The ACCOUNT module stores all of the user's data. It has methods for retrieving and updating a user's profile; this includes items like the user's profile picture, profile description, list of interests, gender, and birthdate.

The next module we have is ADMINISTRATION. This module defines what administrator accounts are authorized to do. This module defines functions for things like listing ALL site users, banning users, reading a user's chat logs to determine if they've been behaving inappropriately, or viewing a user's profile. This module is necessary to make sure ChatSchoolette remains a clean and fun environment for students to make new friends.

The AUTHORIZATION module defines all of the functions for determining if a user really is who he or she says they are. This module has functions for logging users in and out, resetting lost passwords, and loading user's into sessions so cookies can be tracked for functions like "Remember Me."

The last module is CHAT. Unsurprisingly, ChatSchoolette relies heavily upon the CHAT module to function well as a website. The CHAT module defines the functions for users to request a chat partner, list their chatting preferences, and (obviously) connecting to a chat session.

All of our modules are tied together by the Python-Flask framework. If you've never seen Flask before, it basically intuitively ties everything in a project together as long as the individual Python files are defined within a module. As I just outlined before this, this is what we've done. As a result, any change to an individual submodule is automatically propagated throughout the site with no other changes required within the code. Because we are using Python-Flask, we never have to have the modules talk to each other. Moreover, our components should NOT interact with each other directly, EVER. Any resource needed by a module should be retrieved directly from the database or through the Flask API. This is the point of the modularization of the project.

We use SQLAlchemy for our database operations. This is an incredibly powerful package that automatically defines all INSERT, UPDATE, DELETE, etc. queries for you as long as you define classes that extend "db.Model". This leads us to not ever having to write or rewrite SQL code as all changes are automatically picked up by SQLAlchemy. Furthermore, if we ever want to move from SQLITE to another database, we just redefine the Database URI and SQLAlchemy can migrate everything for us.

Our group used bottom-up testing to develop ChatSchoolette. We wanted to get the individual sub-modules and methods working before we integrated them into the site as a whole. Since websites are relatively simple things to code (compared to the actual algorithms USED by the website), we weren't too worried in making sure that things like loading "/index.html" was loading a page in the correct format for the user. It was much more important for us to ensure that the individual functions that created, retrieved, updated, and deleted our data were working correctly and efficiently. Because of this, we decided to write all of our test cases from a bottom-up style of testing.

MODULE: ACCOUNT

Incremental Testing

#	Description	Severity	How to Correct
1	Users cannot see their list of friends	3	Add a route to show the user's friends list
2	User has no way to send a message to a friend	2	Add a route to message a friend and send the message to the friend's inbox
3	User has no notification of new messages in their inbox	2	Add a button on the navbar to show users they have unread messages

Regression Testing

#	Description	Severity	How to Correct
1	Adding friends list button caused other buttons on account page to have weird unappealing offset	3	Change css styling to fix button offsets
2	Similar issue with adding unread messages -- button offset got messed up	3	Change css styling to accommodate unread messages button
3	Unread message notification never disappears	2	Add a field to the messages table to mark whether or not the message has been read

MODULE: ADMINISTRATION

Incremental Testing

#	Description	Severity	How to Correct
1	Admins have no way of viewing messages between friends (currently only set up to view messages in a chat session)	1	Extend the "view messages" function to see P2P messages

2	Admin can’t send the message to user since they are not friend each other	1	Grant admin a privilege to send a message to any user
3	Banned users can simply recreate their account	2	Change the method of banning to save the user’s account but list them as banned (add field to db)

Regression Testing

#	Description	Severity	How to Correct
1	After adding feature to view P2P messages, sometimes the user’s chat session messages would not appear.	2	Separate the data types so there is no risk of data being overwritten when loading user messages (avoid race condition)
2	After giving privilege to admin that admin can send a message to user, but then user can reply random/spam message back to admin.	1	When sender is admin, block the reply button on the message
3	Banned users can login, but don’t realize they have been banned	3	Show a message to banned users alerting them of their ban and why they were banned

MODULE: AUTHORIZATION

Incremental Testing

#	Description	Severity	How to Correct
1	**No Changes were made to the authorization module during this sprint**		

Regression Testing

#	Description	Severity	How to Correct

1	**No Changes were made to the authorization module during this sprint**		

MODULE: CHAT

Incremental Testing

#	Description	Severity	How to Correct
1	If user type more than 140 characters, server would only send the first 140 characters	2	Change our database schema. Change datatype from string to text for longer text storing.
2	Justification and other specifications needed for user-friendly and readable messages	2	Format text using CSS to allow for justified text and proper attributes such as size, style, line height.
3	ASCII limits things that we can type.	3	Change it to UTF-8

Regression Testing

#	Description	Severity	How to Correct
1	Adding styles for button widths in CSS , with lack of foresight of adding or removing buttons caused margin issues.	3	Format buttons by adding more CSS to allow for modifiability to page without future style issues.
2	Changing database schema from string(140) to text gave compatibility issue with existing data	2	Existing database had to be converted.
3	When adding video module to chat, text box would not appear correctly	2	Use more CSS witchcraft to make things line up correctly