Design Document

High Level Design

Summary:

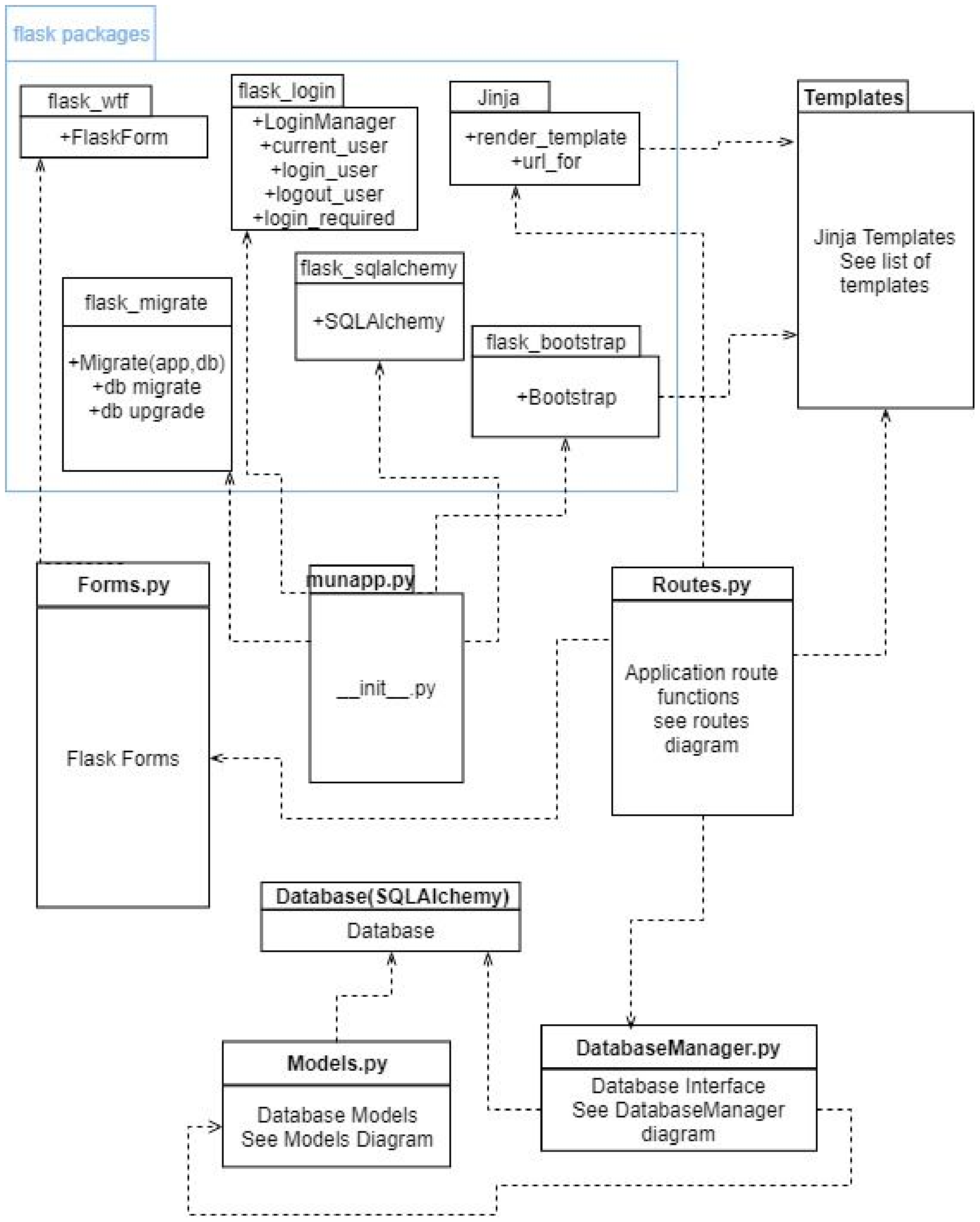
The application was created using the flask framework, a standardized framework that integrates multiple packages and engines to allow python code be used for programming and creating web applications. Flask's main interface is with jinja, a template engine that allows for html code to be written in a way that can interpret arguments provided by python code, and render templates through python function routes. Several main modules were developed by the project team including DatabaseManager (a database interface tool), Routes (the routing tool that checks what functions to run when webpages are requested), Forms (collection of forms used in the application) and Models (the database Classes (known as models) used for the application). These modules are described and broken down in greater detail in the Module Design and Interfaces section, including what packages and modules each interface with and the purpose of each module.

Weaknesses:

- Some modules and libraries were added to the application design later in development (for example, bootstrap), so the full strength of these modules are not realized and may not be integrated in the design as well as features that were developed early in the development and design cycle (such as DatabaseManager).
- Modules and functions were initially developed while gaining proficiency in the various packages offered through flask, python, etc., so some earlier code may be inefficient or may not completely match the overall high level design
- The high level design evolved greatly over the development of the project, increasing in complexity from the original overview. As seen in the high level design document, this caused greater overall complexity which reduces supportability and portability of the overall application for future releases. Further work on this project would see the design revised once more, and modules redesigned to support a cleaner overview.
- Some of the modules (e.g. routes.py) are large in scope, and would be more understandable if broken down into smaller modules that take care of more similar tasks. This scope grew quickly over the course of the project and was not able to be implemented as part of the initial design, but is detailed in the 'Next Steps' as an area of improvement.

See diagram on next page -

Diagram:



flask packages
flask_wtf
+FlaskForm
flask_login
+LoginManager
+current_user
+login_user
+logout_user
+login_required
Jinja
+render_template
+url_for
Templates
Jinja Templates
See list of templates
flask_migrate
+Migrate(app,db)
+db migrate
+db upgrade
flask_sqlalchemy
+SQLAlchemy
flask_bootstrap
+Bootstrap
Forms.py
Flask Forms
munapp.py
__init__.py
Routes.py
Application route functions
see routes diagram
Database(SQLAlchemy)
Database
Models.py
Database Models
See Models Diagram
DatabaseManager.py
Database Interface
See DatabaseManager diagram

Module Design and Interfaces

This section will describe the various modules and interfaces used and developed for this application. Each module includes a brief overview of its functionality, as well as an in depth list of all the functions, methods, and classes included with the module

Flask Packages:

Summary:

Multiple flask packages were imported and utilized in this application as the main framework that modules were built around. The particular packages used are listed under ***Classes and Functions.***

Classes and Functions: Flask, flask_sqlalchemy (SQLalchemy), flask_migrate (Migrate), flask_login (LoginManager), flask_bootstrap (Bootstrap), flask (render_template, flash, redirect, url_for, request, g)

Diagram:

See high level design document, in 'Flask Packages' section.

Forms:

Summary:

Flask forms are created using the Flask WTForms framework, which simplifies creation of Flask Forms and allows easy integration with Bootstrap for content formatting. The Forms.py module interfaces with flask_wtf to retrieve the FlaskForm framework, and also imports the various field types from wtforms (such as StringField, PasswordField, etc.). The various forms used in the application are detailed in the **Class and Function Table**.

Current Dependencies (imported functions): flask_wtf (FlaskForm), wtforms (StringField,PasswordField, BooleanField, SubmitField, TextAreaField), wtforms.validators (ValidationError, DataRequired, Email, EqualTo, Length), Models (User, Group)

Weaknesses:

- Reliance on a large number of forms that do very similar things, just with different labelling and names, creates a larger overhead of classes to maintain. While these distinctions assist with creating clarity in what forms are able to do what, it creates extra work for the programmer who must maintain the program.
- Lack of modularization in some functions, due to initial program code design of the project. Can be easily remedied by moving functions to Routes, and is not reflected in the high level design.

Class and Function Table:

Classes/Function and Arguments	Description
RegistrationForm(FlaskForm)	Form used for registration. Requires a username, email, password, password confirmation, and submission.
LoginForm(FlaskForm)	Form used for login. Requires a username, password, and submission. Also has a optional "Remember Me" field to remember the user.
TopicForm(FlaskForm)	Form used to create a topic. Requires a title, topic body, and submission.
CommentForm(FlaskForm)	Form used to create a comment. Requires a comment body, and submission.
GroupForm(FlaskForm)	Form used to create a group. Requires a group title, and submission.
AddUserForm(FlaskForm)	Form used to add a user to a group. Requires a username and submission.
FindUserForm(FlaskForm)	Form used to find a user. Requires a username.
SubscriptionForm(FlaskForm)	Form used to subscribe to a topic. Requires confirmation by submission.
UnsubscriptionForm(FlaskForm)	Form used to unsubscribe to a topic. Requires confirmation by submission.
LeaveGroupForm(FlaskForm)	Form used to leave a group. Requires confirmation by submission.

routes.py:

Summary:

The routes.py module interfaces with flask libraries, the Database Manager, Forms, and the Jinja templates. It is at the center of the application and is where most functionality flows through to achieve the necessary view functions.

Current Dependencies (imported functions): DatabaseManager, Forms, flask_login (current_user, login_user, logout_user, login_required), flask(render_template, flash, redirect, url_for, request, g)

Weaknesses:

- Many functions were written early on in and had addendums by several group members, resulting in some bloated functions that, while functional, can be cumbersome to understand their functionality. Creating helper functions or reevaluating current functions to find optimizations would be beneficial.
- Function calls are not always consistent across route functions. Standardizing whether an object or an object field (e.g. topic id vs a topic object) is passed to a webpage would make future development and design easier and clearer.
- Underutilization of modularization, as many functions are stored in this module. As detailed in the high level design weakness, plan to further modularize this in future iterations.

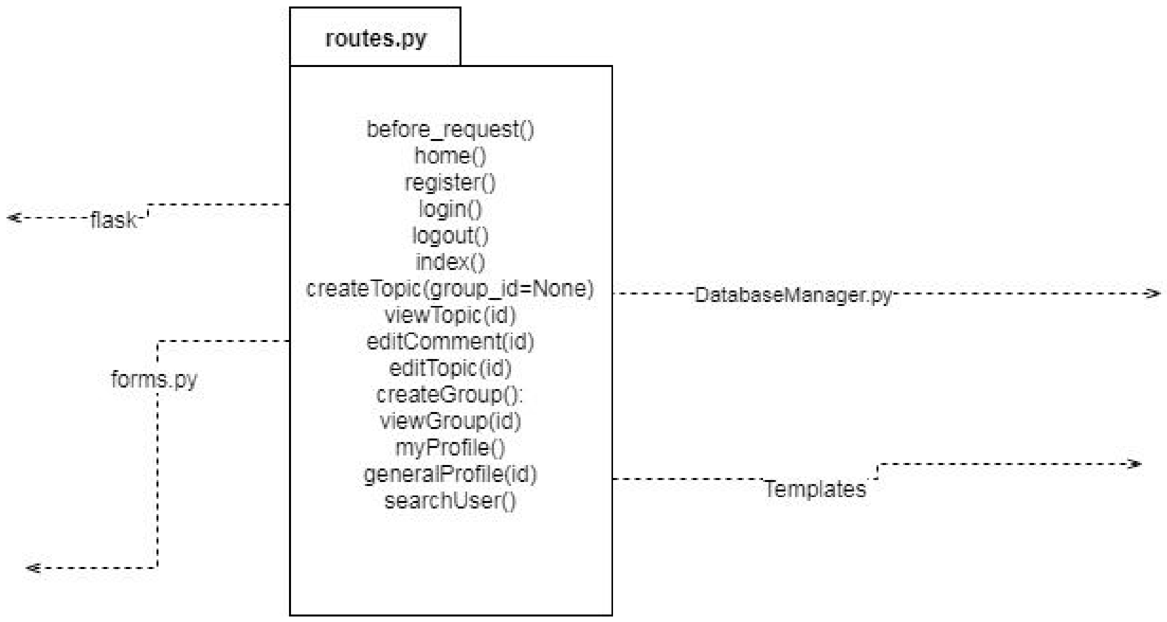
Class and Function Table:

Function Name	**Description**
before_request()	Contains functions to be run before each request by the application. Currently, checks if the current user is authenticated (ie. Logged in) and if so assigns a FindUserForm to g. This is used in the base.html template for searching user profiles.
home()	Function related to home, decorated by app route '/' and '/home'. Checks if the current user is authenticated and if so, redirects them to the index function. Otherwise, renders the home.html file with a login and registration form.
register()	Function related to registration, decorated by app route '/register'. Checks if the current user is authenticated and if so, redirects them to the index function. Otherwise, renders the register.html template with a registration form. On submission, validates the data and if valid, attempts to register the in (interfacing with DatabaseManager)
login()	Function related to login, decorated by app route '/login'. Checks if the current user is authenticated and if so, redirects them to the index function. Otherwise, renders the login.html page with a login form. On submission, validates the data and if valid, attempts to log the user in (interfacing with DatabaseManager)
logout()	Function related to logout, decorated by app route '/logout'. Returns the DatabaseManager.logout() function.
index()	Function related to the index, decorated by app route '/index'. Interfaces with database manager to get public topics,
createTopic(group_id=None)	Function related to creating a topic, taking the argument group_id (or assigning it None if no id is supplied), decorated by app route '/create_topic' and '/create_topic/<group_id>'. Renders the create_topic html page with a TopicForm. On submission of the form, checks for validation and returns the DatabaseManager.addTopic function (Creating a new topic). Redirects to the viewTopic function with the new topic id as the argument.
viewTopic(id)	Function related to viewing a topic, taking the argument id as the topic to view. Decorated by the app.route '/post/<id>/. Retrieves the topic from the database using the topic id. Then checks if the topic is associated with a group. If so, checks if the current user is a member of the group, if they are, checks if they are currently subscribed to the topic and renders the post.html page with the topic, comment form, and sub or unsub btn (depending on subscription status) as arguments. Otherwise, returns the user to the home page and flashes an error message. If the topic does not have an associated group, the group membership check is skipped and other functions described above proceed as normal.
editComment(id)	Function related to editing a comment, taking the argument 'id' as the comment to edit. Decorated by the app.route '/edit_comment/<id>'. Checks if the user is the comment author by calling DatabaseManage.checkCommentAuthor. If returns true, renders the edit_comment.html page with a CommentForm prefilled with the

	current comment body. If returns false, redirects the user to the home function and flashes an error message.
editTopic(id)	Function related to editing a topic, taking argument 'id' as the topic to edit. Decorated by the app.route '/edit_topic/<id>'. Creates a Topic Form and gets the topic object from the Database Manager using the id. It validates if the current user is the topic author, if not, they are redirected to the home function and flashed and error message. If they are, the edit_topic.html page is rendered with the topic form and topic object as arguments.
createGroup():	Function related to creating a group. Decorated by the app.route '/create_group/'. Creates a Group Form and renders the template of the create_group.html page using the form as an argument. Validates data on submission, creating a group with the supplied data as its title.
viewGroup(id)	Function related to viewing a group, taking argument 'id' as the group id to be viewed. Decorated by the app.route '/group/<id>'. Loads the group object from the database using the Database Manager and id as argument. Checks if the current user is a member of the group, and if not, redirects them to the home function and flashes an error message. If they are, generates the group.html template with the AddUserForm for adding members to the group, and LeaveGroupForm for confirmation of leaving a group.
myProfile()	Function related to viewing the current users profile. Decorated by the app.route '/my_profile. Retrieves the users subscriptions, notifications, comments, topics, and groups from the database and renders the my_profile.html template.
generalProfile(id)	Function related to viewing a user's profile. Decorated by the app.route '/general_profile. Retrieves the requested users comments and topics from the database and renders the my_profile.html template.
searchUser()	Function related to searching for a user profile. Loads the profile_search form from the g variable. If valid on submission, retrieves the given user by username from the database. If 'None', redirects the user to the home function and flashes an error message. If not 'None', redirects the user to the generalProfile function with the found user's id as argument.

See diagram on next page -

Diagram:



DatabaseManager:

Summary:

The DatabaseManager.py module provides the boundary for other modules so that they do not have to deal with the database or database models directly. This interface is used primarily to query the database and add, create, edit, or remove entries as desired. DatabaseManager primarily provides functionality to Routes.py, and utilizes the database models of Models.py along with connection the database directly. This was designed so that if the persistence model changed, details of the rest of the application would remain the same while the lower level functionality of DatabaseManager could be easily ported.

Current Dependencies (imported functions): models (all models), flask_login(current_user, login_user, logout_user), database

Weaknesses:

- Standardization of getting and setting methods across the interface. While the interface is standardized for its external needs (ie. routes reliance on the database manager to get and set attributes), internally it uses a mix of ids and the actual objects as arguments, and does not always use internal helper functions set up for standard retrieval of objects. This is due to the different approaches of the programmers on this project as well as the evolution of the project, and could be fixed with more time and attention to standardizing the internal function of the interface.

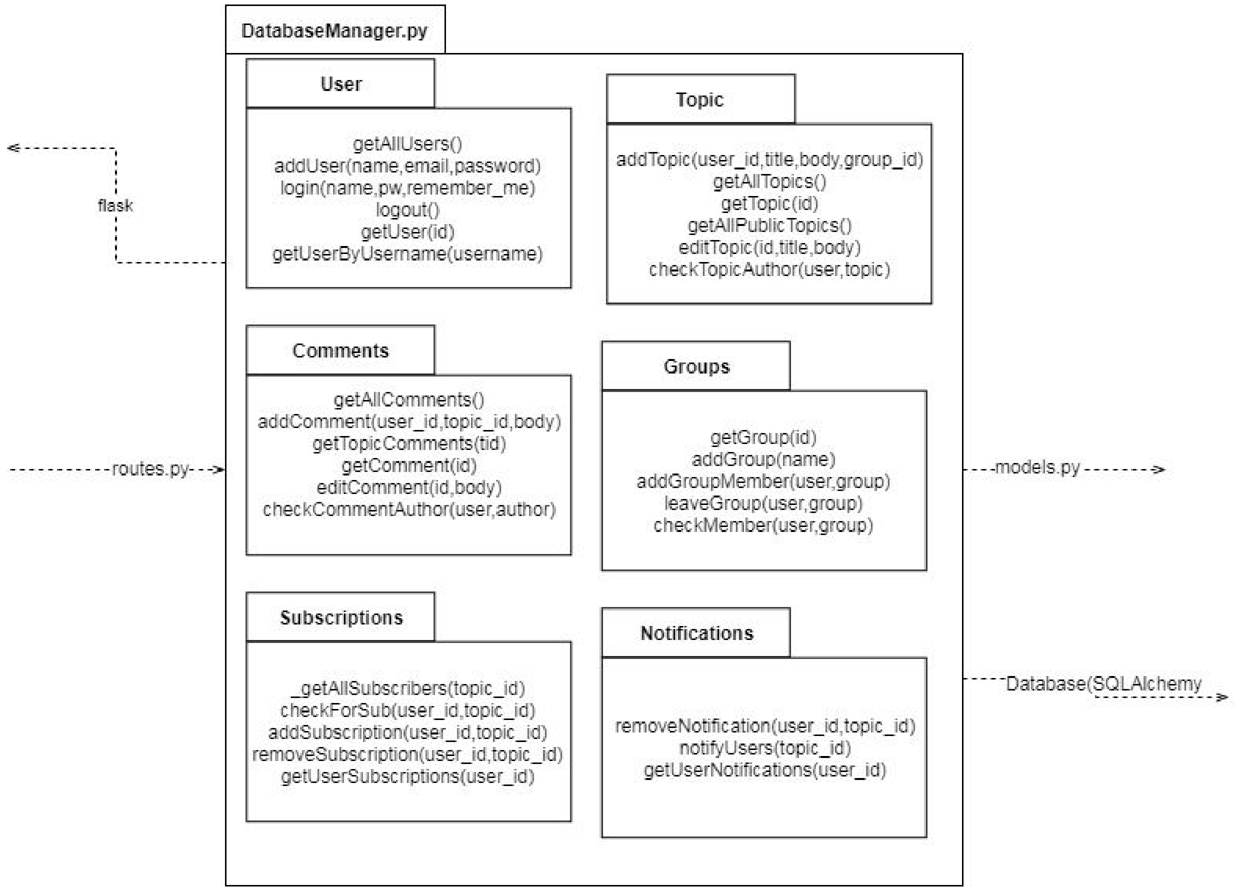
Class and Function Table:

Function Name	**Description**
removeNotification(user_id, topic_id)	Queries the database for a user and topic using their respective ids, and removes the notification for the given topic and given user and commits changes to the database
notifyUsers(topic_id)	Calls getAllSubscribers for the given topic id and gets the topic object for a topic id.
_getAllSubscribers(topic_id)	Queries the database for a topic given a topic id as argument. Returns the topic.subscribers field for that topic.
getUserNotifications(user_id)	Queries the database for a user with given user id as argument, returns the user.notifications field for that user.
getAllUsers()	Queries the database User model and returns all users.
getAllComments()	Queries the database Comment model and returns all comments.
addUser(name, email, password)	Creates a new user object using a given name, email, and password, and commits to the database.
login(name, pw, remember_me)	Checks the username and password supplied and, if valid, logs the user in to their account.
logout()	Logs the user out.
addTopic(user_id, title, body, group_id, public=True)	Adds a topic with the user_id as author, title as topic title, body as the topic content, and group_id if it is associated with a group (otherwise it is set to None). Commits these changes to the database
addComment(user_id, topic_id, body)	Adds a comment with user_d, topic_id, and body. Commits these changes to the database
getAllTopics()	Queries the database for all topics.
getTopicComments(tid)	Queries the database for all comments for a given topic id. Returns topics
getTopic(id)	Queries the database to retrieve a topic with the given id. Returns topic.
getComment(id)	Queries the database to retrieve a comment with the given id. Returns comment.
editComment(id, body)	Retrieves a comment from the database using getcomment(id), and replaces the comment body with the supplied body argument. Commits these changes to the database
editTopic(id, title, body)	Retrieves a topic from the database using getTopic(id), and replaces the topic body and title with the supplied body and title arguments. Commits these changes to the database
getGroup(id)	Queries the database to retrieve a group with the given id. Returns group.
addGroup(name)	Creates a Group model object with name as its name attribute. Adds the current user to the list of group members, and
addGroupMember(user,group)	Appends a given user to the group.members relationship for the specified group and commits to the database.

getUser(id)	Queries the database to retrieve a user with the given id. Returns user.
getUserByUsername(username)	Queries the database to retrieve a user with the given username. Returns user (or None if doesn't exist)
checkMember(user,group)	Queries the database to see if a given user is in the group.members list. Returns true if they are, false otherwise.
getAllPublicTopics()	Queries the database for all topics that have 'None' assigned to their group_id. Returns topics.
checkTopicAuthor(user,topic)	Checks if if the user.id of the user is equivalent to the to topic.author.id of the topic. Returns True if so, False otherwise.
checkCommentAuthor(user,comment)	Checks if if the user.id of the user is equivalent to the to comment.author.id of the comment. Returns True if so, False otherwise.
checkForSub(user_id, topic_id)	Queries the database for the given user id and topic id to get the user and topic. Checks if the topic is in the user.subscriptions, returns True if so, returns False otherwise.
addSubscription(user_id, topic_id)	Queries the database for the given user id and topic id to get the user and topic. Appends the topic to the users subscription relationships, and commits to the database.
removeSubscription(user_id, topic_id)	Queries the database for the given user id and topic id to get the user and topic. Checks if the topic is in the user.subscriptions, and removes if it is and commits to the database.
getUserSubscriptions(user_id)	Queries the database for a given user id and returns their subscriptions (user.subscriptions)
leaveGroup(user,group)	Removes the given user from the group.members list for the given member and commits to the database.

See diagram on next page -

Diagram (note: simplified into areas for easier understanding (e.g. getAllUsers() put under Users heading) , functions are not categorized this way in the code):


DatabaseManager.py
User
getAllUsers()
addUser(name,email,password)
login(name,pw,remember_me)
logout()
getUser(id)
getUserByUsername(username)
Topic
addTopic(user_id,title,body,group_id)
getAllTopics()
getTopic(id)
getAllPublicTopics()
editTopic(id,title,body)
checkTopicAuthor(user,topic)
Comments
getAllComments()
addComment(user_id,topic_id,body)
getTopicComments(tid)
getComment(id)
editComment(id,body)
checkCommentAuthor(user,author)
Groups
getGroup(id)
addGroup(name)
addGroupMember(user,group)
leaveGroup(user,group)
checkMember(user,group)
Subscriptions
_getAllSubscribers(topic_id)
checkForSub(user_id,topic_id)
addSubscription(user_id,topic_id)
removeSubscription(user_id,topic_id)
getUserSubscriptions(user_id)
Notifications
removeNotification(user_id,topic_id)
notifyUsers(topic_id)
getUserNotifications(user_id)
flask
routes.py
models.py
Database(SQLAlchemy

Models:

Summary:

The models.py module defines the classes used for tables used in the SQLalchemy database. As defined in the class and function table, each Class has attributes that are stored and relationships created for linkages to other class to establish connections as necessary.

Current Dependencies (imported functions): flask_login (UserMixin), datetime, database

Weaknesses:

- Database design evolved over the course of the project, therefore some fields are no longer used and methods of connections could be standardized (e.g. relationship types).

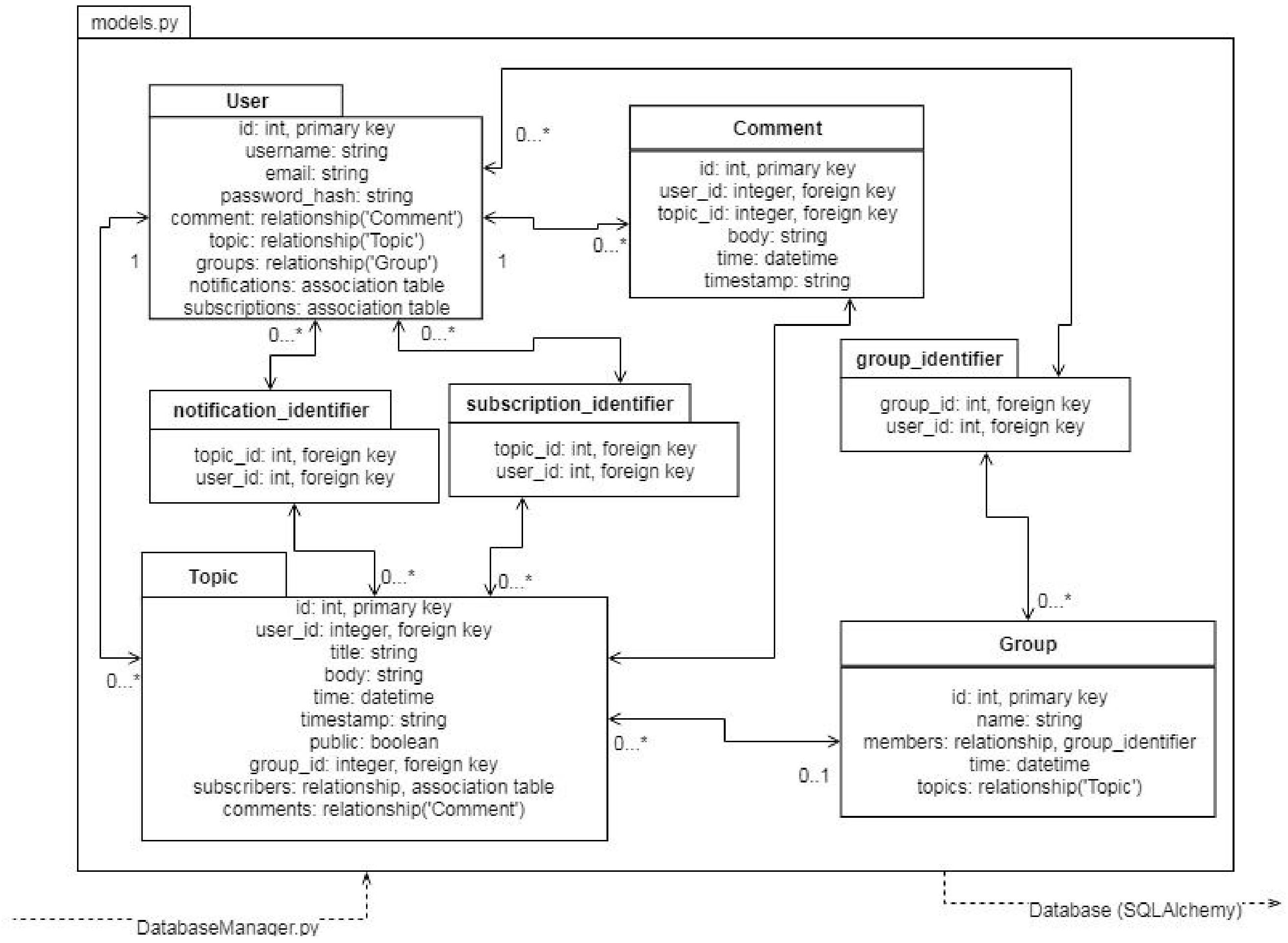
Class and Function Table:

Classes	**Description**
User(UserMixin, db.Model):	Database model (Class) used for topics. Contains the fields: ● id = integer, primary key automatically assigned to a user record on creation ● username = string, username supplied by the user at time of creation (registration) ● email = string, email supplied by the user at time of creation (registration) ● password_hash = string, generated by the password supplied by the user at time of creation (registration) ● comment = relationship, comments that are linked to the user (the user has created) ● topic = relationship, topics that are linked to the user (the user has created) ● groups = relationship, groups that are linked to the user (the user is a member of) ● notifications = relationship, references the notification_identifer table to retrieve users who have notifications for posts on topics they are subscribed to
Topic(db.Model)	Database model (Class) used for topics. Contains the fields: ● id = integer, primary key automatically assigned to a topic record on creation ● user_id = integer, foreign key that links the topic to a user to record the topic author ● title = string, title of the topic assigned by the user at time of creation ● body = string, body of the topic assigned by the user at time of creation ● time = datetime, timestamp of topic creation ● timestamp = string, assigned time of creation ● group_id = relationship, foreign key that links the topic to a group or None if not linked to a group ● subscribers = relationship, references the subscription_identifer table to retrieve users who are subscribed to a topic (matched topic_id's and user_id's) ● comments= relationship, reference to comments that are linked to the topic
Comment(db.Model)	Database model (Class) used for comments. Contains the fields: ● id = integer, primary key automatically assigned to a comment record on creation ● user_id = integer, foreign key that links the comment to a user to record the comment author ● topic_id = integer, foreign key that links the comment to a topic to record the associated topic thread for the comment

	● body = string, content of the comment supplied by the user at time of creation ● time = datetime, timestamp of comment creation ● timestamp = string, assigned time of creation
Group(db.Model)	Database model (Class) used for groups. Contains the fields: ● id = integer, primary key automatically assigned to a group record ● name = string, name of group assigned by user at time of creation ● members = relationship, references the group_identifier table to retrieve users who are in the group (matched group_id's and user_id's) ● time = datetime, timestamp of group creation ● timestamp = string, assigned time of creation ● topics = relationship, store of the topics that are associated with the group record
group_identifier	Association table used to add a many to many relationship between users and groups. Contains the user id and group id of associated entries
subscription_identifier	Association table used to add a many to many relationship between users and topics for subscription purposes. Contains the user id and topic id of associated entries
notification_identifier	Association table used to add a many to many relationship between users and topics for notification purposes. Contains the user id and topic id of associated entries

See diagram on next page -

Diagram:


models.py
User
id: int, primary key
username: string
email: string
password_hash: string
comment: relationship('Comment')
topic: relationship('Topic')
groups: relationship('Group')
notifications: association table
subscriptions: association table
Comment
id: int, primary key
user_id: integer, foreign key
topic_id: integer, foreign key
body: string
time: datetime
timestamp: string
notification_identifier
topic_id: int, foreign key
user_id: int, foreign key
subscription_identifier
topic_id: int, foreign key
user_id: int, foreign key
group_identifier
group_id: int, foreign key
user_id: int, foreign key
Topic
id: int, primary key
user_id: integer, foreign key
title: string
body: string
time: datetime
timestamp: string
public: boolean
group_id: integer, foreign key
subscribers: relationship, association table
comments: relationship('Comment')
Group
id: int, primary key
name: string
members: relationship, group_identifier
time: datetime
topics: relationship('Topic')
0...*
1
0..1
DatabaseManager.py
Database (SQLAlchemy)

Templates:

Summary:

HTML templates for the application are created using the Jinja2 template engine, which is designed for python. Additionally, flask bootstrap is implemented in the template, to allow additional control and convenience of formatting.

Current Dependencies (imported functions): Bootstrap, Bootstrap-wtf

Weaknesses:

- As flask bootstrap was imported well into the development of the project, all pages are not fully standardized using bootstrap formatting. Further development would include redesign of the each html template to assure full compliance and consistency across template styles.

List of templates: 401.html, 404.html, 500.html, base.html, create_group.html, create_topic.html, edit_comment.html, edit_topic.html, error_base.html, general_profile.html, group.html, home.html, index.html, login.html, my_profile.html, post.html, register.html, view_topic.html