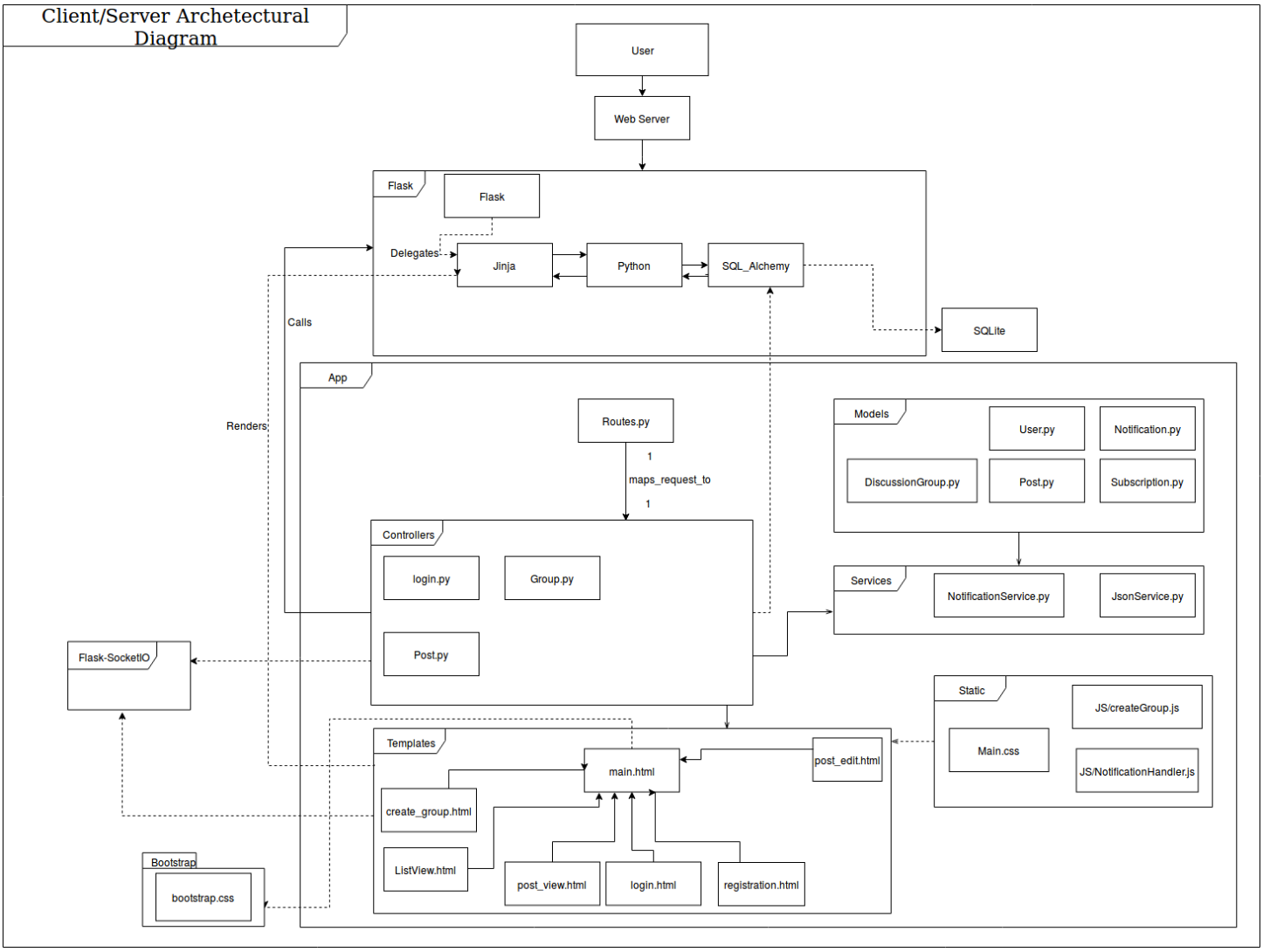
Explaination of Archetectural Structure of the Prototype:

For the architectural structural of the prototype, the client/server architectural design is best suited. It separates pieces of the system that needs to use a particular function (the client) from parts of the system that provides those functions (the server). This allows for decoupling between the client (the user) and the server, allowing developers to advance the prototype in the future

How the Client/Server architectural structure works, goes as follows:

- The User sends a web request to the Web Server.
- The web request is sent to the python package which executes the request and retrieves the data needed from the data base and the Jinja template.
- The python package has routes which connects the modules i.e.
    1. **login.py which is called by routes.py and renders template from login.html jinja template**
    2. **Post.py which is called by routes.py and renders template from postView.html jinja template**
    3. **Subscription.py which is called by routes.py and renders template from Subscription.html jinja template**
    4. **DiscussionGroup.py which is called by routes.py and renders template from DiscussionGroup.html jinja template**
    5. **UserProfile.py which is called by routes.py and renders template from UserProfile.html jinja template**

  With all Jinja files placed in a template created as shown in the uml diagram
- The route.py has functions that enables the web server to provide a requested service for the user.
- The SQLAlchemy python code file, Session is established to connect with the database and represents for all objects you've loaded or associated with it during the lifespan. the Session includes add() method, commit() method, delete() method. add() is use to place instances in the session. commit() is use to write changes to the database. delete() is use to place an instance into the Session's list of objects to be marked as deleted.
- Controllers consists of the python code for each of the requirements. The routes.py file imports the methods for each of the GET and POST requests that the users asks for. This is in a folder colled Controllers and placed in the apps folder.
- Flask delegates jinja to render all templates to extend the main.html, which uses a base design for all the requirements template.
- Flask delegates the backend routes between python, jinja and SQLAlchemy.
- Bootstrap is used to help build the user interface of the prototype. It uses CSS or Cascading Style Sheet to create the design of the prototype.


Client/Server Archetectural Diagram
User
Web Server
Flask
Flask
Delegates
Jinja
Python
SQL_Alchemy
SQLite
Calls
App
Renders
Routes.py
1
maps_request_to
1
Models
User.py
Notification.py
DiscussionGroup.py
Post.py
Subscription.py
Controllers
login.py
Group.py
Post.py
Services
NotificationService.py
JsonService.py
Flask-SocketIO
Static
JS/createGroup.js
Main.css
JS/NotificationHandler.js
Templates
main.html
post_edit.html
create_group.html
ListView.html
post_view.html
login.html
registration.html
Bootstrap
bootstrap.css

Explaination of how User Profile works:

The User profile system consists of:

- Database User table.
- Forms use and tools for viewing user information and direct messages
- A pluggable backend system

The system is using SQLAlchemy to access and manage data in the SQLite database. SQLAlchemy is adapted into pythonic domain language.

In the SQLALchemy python code file, Session is established to connect with the database and represents for all objects you've loaded or associated with it during the lifespan. The Session includes add() method, commit() method. add() is use to place instances in the session. commit() is use to write changes to the database.

Controllers consists of the python code of the user proflie. Those forms are using Flask as a microframework, which is using jinja as a template engine.

In the Flask code, the url_for() function will build a Url for a specific function. The render_template() function will render a HTML file which base on the jinja and html code. The get_flashed_message() passes a message of the next request, which generally is a template.

The templates folder consists of main.html and user_Profile.html which are base on jinja and html code. The CSS of the html files will be using the bootstrap library.

Also, the user profile system is using the services package, which can pop up a notification when a user receive a direct message.

Controllers file, routes.py, templates file are all included in app file.

Detailed Description for the High-Level design
User Profile

The system is using SQLAlchemy to access and manage data in the SQLite database. SQLAlchemy is adapted into pythonic domain language.

Template package

The template package includes two html files, which are UserProfile.html and Main.html. those files are all written in html and using jinja to add control statements including url_for(), render_template():

-url_for() will build a url for a specific function.

-render_template() will render a HTML file which base on the jinja and html code.

In this Templates package, we use ninja's template inheritance feature, which allows us to move the parts of the page layout that are common to all templates and put them in a main template from which all other templates are derived. Also, those templates will handle message by using MessageHandler in the user profile **js package .**

Controllers package

The Controllers package consists of three parts of the system, which are Direct Messaging and User Profile. Those are written in python programming language.

Direct Messaging has a post() method and a url_for() method:

-The post() method tells the server that it wants to post some new information to that url and that the server must ensure the data is stored and only stored once.

-url_for() method will redirect a user to another webpage.

User Profile has a get() method:

-The get() method will call the getchat() method in Services package and return the history chat of a user.

The **routes.py package** file imports the methods for each of the GET and POST requests that the user asks for. Also, in this package, it will be using Flask-SocketIO package to give flask applications access to low latency bi-directional communication between the clients and the server.

Sevices package

This package is use to dispatch a message from a user to another user. Also, the getchat() method will return a history chat of a user.

-dispatch(message) method will dispatch a message from a user to another user
-getchat(userId) method will get the history chat below the user information.

SQLALchemy

In the SQLALchemy python code file, Session is established to connect with the database and represents for all objects you've loaded or associated with it during the lifespan. the Session includes add() method, commit() method, delete() method.

-add() is use to place instances in the session.
-commit() is use to write changes to the database.
-close() will safely dispose of any remaining session transaction objects associated with the session.

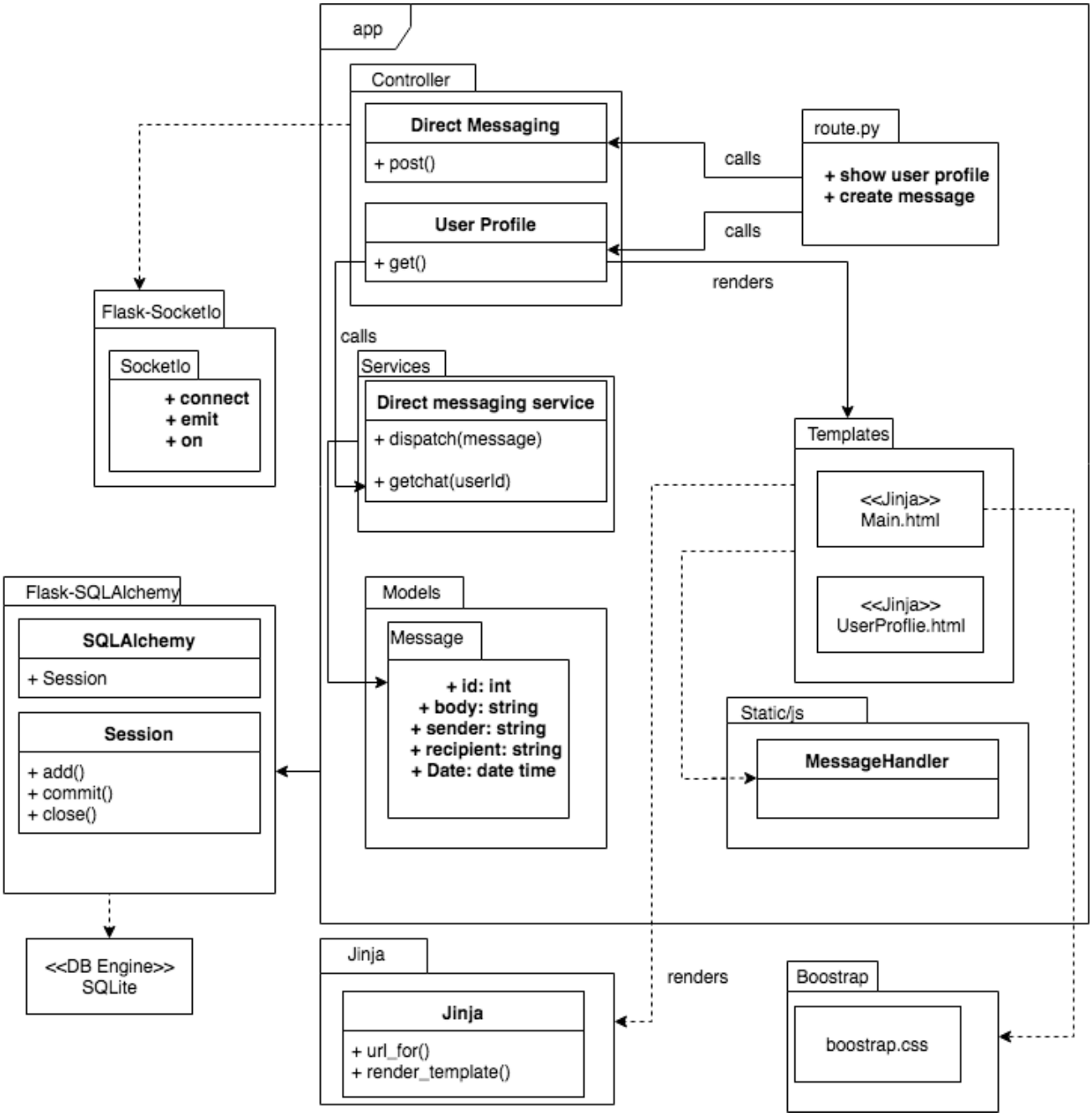
Bootstrap package

The Bootstrap package is use for cascading style sheets. It describes how HTML elements are to be displayed on screen, paper or other media.

Models package

This package includes a message database file. The database file creates a table for a message, which includes:

-id: integer
-body: String
-sender: String
-recipient: String
-Date: date and time


app
Controller
Direct Messaging
+ post()
User Profile
+ get()
route.py
+ show user profile
+ create message
calls
calls
renders
Flask-SocketIo
SocketIo
+ connect
+ emit
+ on
calls
Services
Direct messaging service
+ dispatch(message)
+ getchat(userId)
Templates
<<Jinja>>
Main.html
<<Jinja>>
UserProflie.html
Flask-SQLAlchemy
SQLAlchemy
+ Session
Session
+ add()
+ commit()
+ close()
Models
Message
+ id: int
+ body: string
+ sender: string
+ recipient: string
+ Date: date time
Static/js
MessageHandler
<<DB Engine>>
SQLite
Jinja
Jinja
+ url_for()
+ render_template()
renders
Boostrap
boostrap.css

Explaination of how User Authentication works:

The user authentication system consists of:

- Database user table.
- Forms use and tools for logging in user
- A pluggable backend system

The system is using SQLAlchemy to access and manage data in the SQLite database. SQLAlchemy is adapted into pythonic domain language.

In the SQLALchemy python code file, Session is established to connect with the database and represents for all objects you've loaded or associated with it during the lifespan. the Session includes add() method, commit() method, delete() method. add() is use to place instances in the session. commit() is use to write changes to the database. delete() is use to place an instance into the Session's list of objects to be marked as deleted.

Controllers consists of the python code of Login form, Register form and Logout form. Those forms are using Flask as a microframework, which is using jinja as a template engine.

In the Flask code, the url_for() function will build a Url for a specific function. The render_template() function will render a HTML file which base on the jinja and html code. The get_flashed_message() passes a message o the next request, which generally is a template.

The templates folder consists of main.html, login.html, registration.html, which are base on jinja and html code. The CSS of the html files will be using the bootstrap library.

Controllers file, routes.py, templates file are all included in app file.

Detailed Description for the Low-Level design

Template package

The Templates package includes three html files, which are main.html, login.html, registration.html. Those files are all written in html and using jinja to add control statements, including get_flashed_message(), url_for(), render_template():

-get_flashed_message will passes a message o the next request, which generally is a template.

-url_for() will build a url for a specific function.

-render_template will render a HTML file which base on the jinja and html code.

In this Templates package, we use ninja's template inheritance feature, which allows us to move the parts of the page layout that are common to all templates and put them in a main template from which all other templates are derived.

Controllers package

The Controllers package consists of three parts of the system, which are Login, Logout, Register. Those are written in python programming language.

Login and Register both have a **get()** function and a post() function.:

-The get() function will return a unique identifier for the user.

-The post() function will summit username and user password in the HTTP body.

These two controllers are all using a function called **url_for()** to redirect a user to another webpage. Also, The routes.py file imports the methods for each of the GET and POST requests that the users asks for.

Logout has a **post()** function to tell the server that it wants to post some new information to the URL and that the server must ensure the data is stored and only stored once.

SQLALchemy

In the SQLALchemy python code file, Session is established to connect with the database and represents for all objects you've loaded or associated with it during the lifespan. the Session includes add() method, commit() method, delete() method.

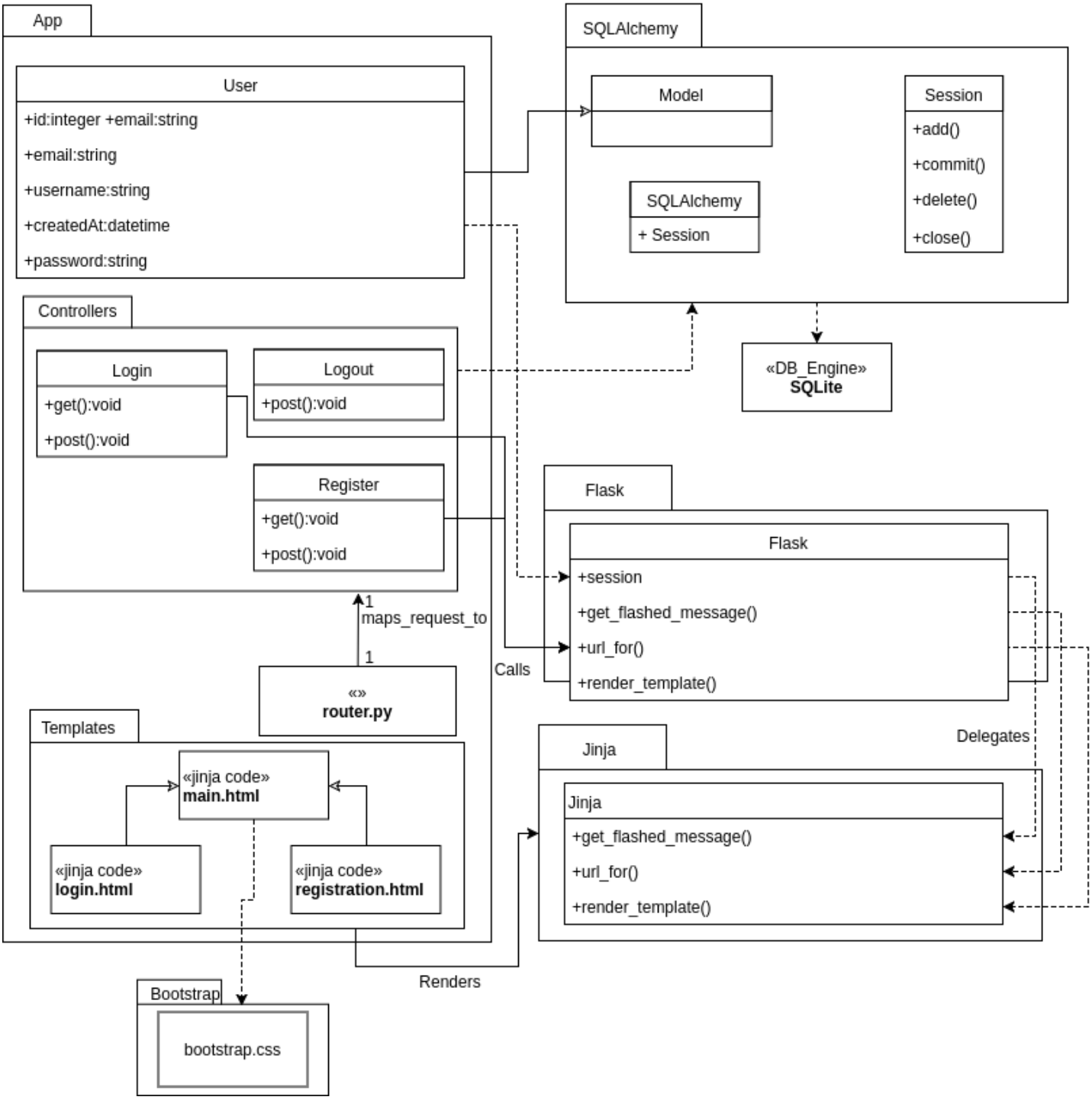
-add() is use to place instances in the session.

-commit() is use to write changes to the database.

-delete() is use to place an instance into the Session's list of objects to be marked as deleted.

Bootstrap package

The Bootstrap package is use for cascading style sheets. It describes how HTML elements are to be displayed on screen, paper or other media.


App
User
+id:integer +email:string
+email:string
+username:string
+createdAt:datetime
+password:string
SQLAlchemy
Model
Session
+add()
+commit()
+delete()
+close()
SQLAlchemy
+ Session
«DB_Engine»
SQLite
Controllers
Login
+get():void
+post():void
Logout
+post():void
Register
+get():void
+post():void
1
maps_request_to
1
«»
router.py
Flask
Flask
+session
+get_flashed_message()
+url_for()
+render_template()
Calls
Templates
«jinja code»
main.html
«jinja code»
login.html
«jinja code»
registration.html
Jinja
Jinja
+get_flashed_message()
+url_for()
+render_template()
Delegates
Renders
Bootstrap
bootstrap.css

Explanation of how Post View works:

The Post View system consists of:

- Database post table.
- Forms use and tools for editing and posting a post
- A pluggable backend system

The system is using SQLAlchemy to access and manage data in the SQLite database. SQLAlchemy is adapted into pythonic domain language.

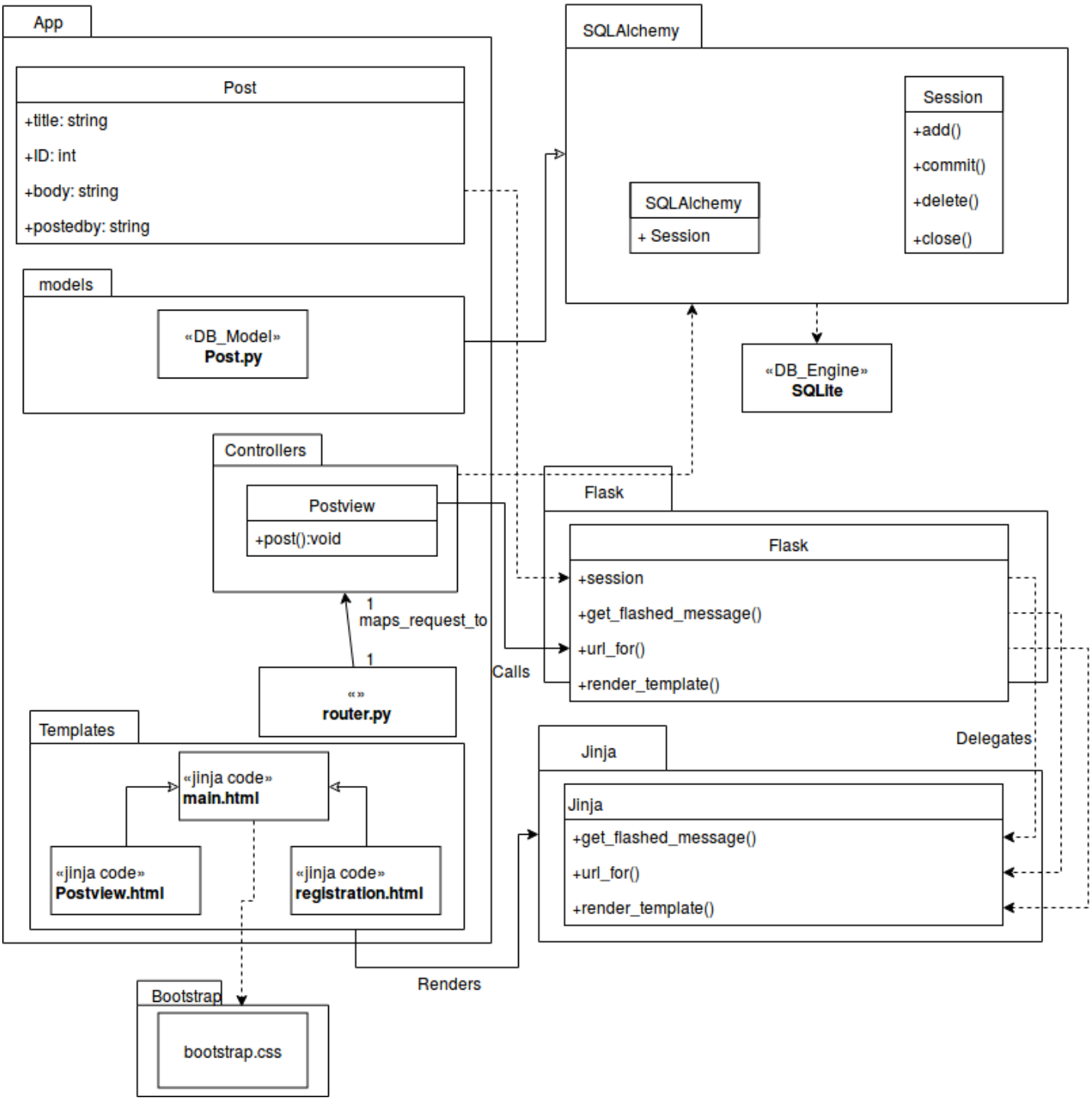
In the SQLALchemy python code file, Session is established to connect with the database and represents for all objects you've loaded or associated with it during the lifespan. The Session includes add() method, commit() method, delete() method. add() is use to place instances in the session. commit() is use to write changes to the database. delete() is use to place an instance into the Session's list of objects to be marked as deleted.

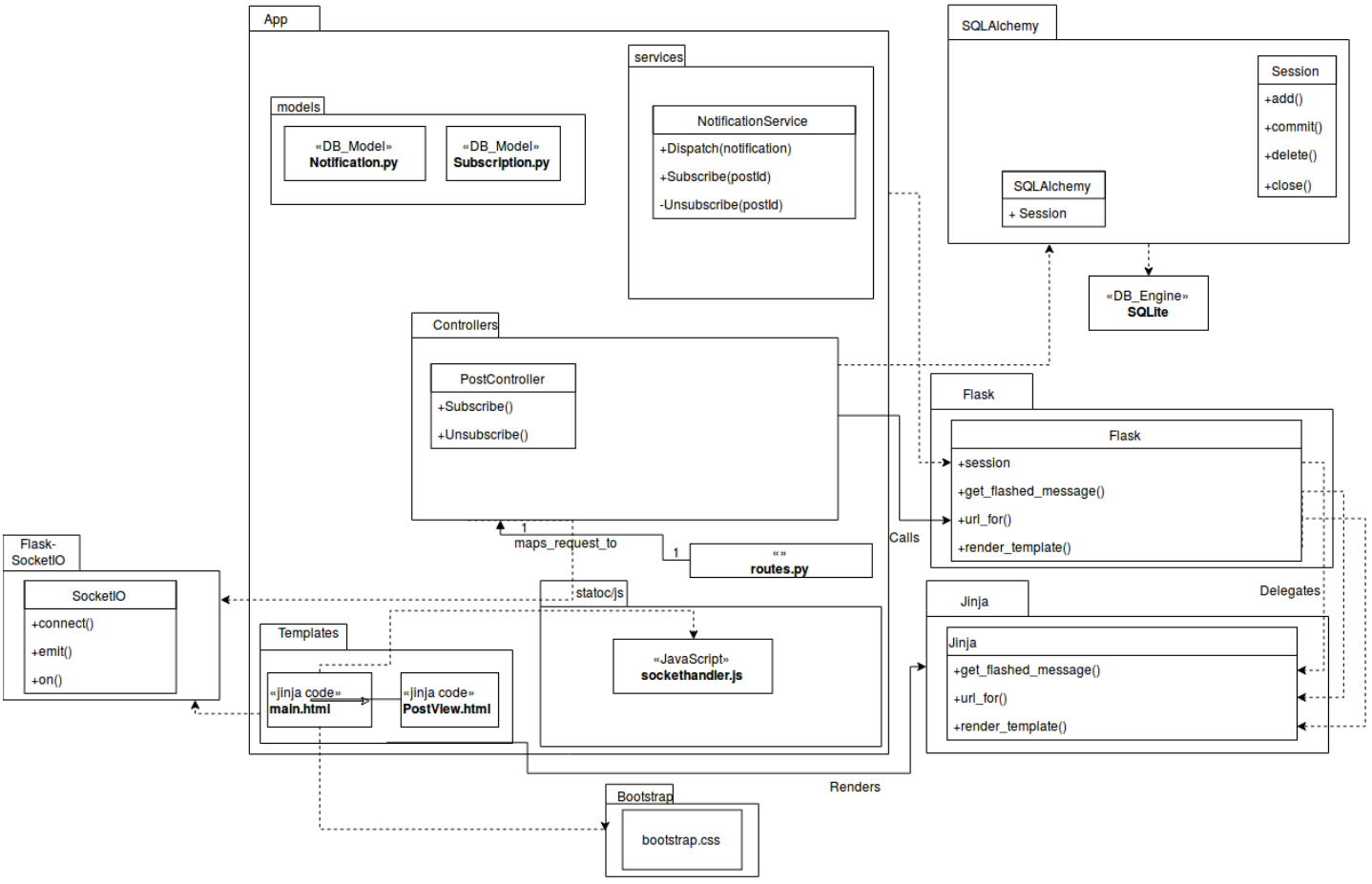
Controllers consists of the python code of the Post editor. Those forms are using Flask as a microframework, which is using jinja as a template engine.

In the Flask code, the url_for() function will build a Url for a specific function. The render_template() function will render a HTML file which base on the jinja and html code. The get_flashed_message() passes a message o the next request, which generally is a template.

The templates folder consists of main.html and PostView.html which are base on jinja and html code. The CSS of the html files will be using the bootstrap library.

Controllers file, routes.py, templates file are all included in app file.


App
Post
+title: string
+ID: int
+body: string
+postedby: string
models
«DB_Model»
Post.py
SQLAlchemy
Session
+add()
+commit()
+delete()
+close()
SQLAlchemy
+ Session
«DB_Engine»
SQLite
Controllers
Postview
+post():void
1
maps_request_to
1
«»
router.py
Flask
Flask
+session
+get_flashed_message()
+url_for()
+render_template()
Calls
Templates
«jinja code»
main.html
«jinja code»
Postview.html
«jinja code»
registration.html
Jinja
Jinja
+get_flashed_message()
+url_for()
+render_template()
Delegates
Renders
Bootstrap
bootstrap.css


App
models
«DB_Model»
Notification.py
«DB_Model»
Subscription.py
services
NotificationService
+Dispatch(notification)
+Subscribe(postId)
-Unsubscribe(postId)
SQLAlchemy
Session
+add()
+commit()
+delete()
+close()
SQLAlchemy
+ Session
«DB_Engine»
SQLite
Controllers
PostController
+Subscribe()
+Unsubscribe()
Flask
Flask
+session
+get_flashed_message()
+url_for()
+render_template()
Calls
1
maps_request_to
1
«»
routes.py
Flask-
SocketIO
SocketIO
+connect()
+emit()
+on()
statoc/js
«JavaScript»
sockethandler.js
Templates
«jinja code»
main.html
«jinja code»
PostView.html
Jinja
Delegates
Jinja
+get_flashed_message()
+url_for()
+render_template()
Renders
Bootstrap
bootstrap.css

Explanation of how Discussion Group works:

The Discussion Group system consists of:

- Database Discussion Group table.
- JS component to search for users to add into a discussion group
- A pluggable backend system

The system is using SQLAlchemy to access and manage data in the SQLite database. SQLAlchemy is adapted into pythonic domain language.

In the SQLALchemy python code file, Session is established to connect with the database and represents for all objects you've loaded or associated with it during the lifespan. The Session includes add() method, commit() method, delete() method. add() is use to place instances in the session. commit() is use to write changes to the database. delete() is use to place an instance into the Session's list of objects to be marked as deleted.

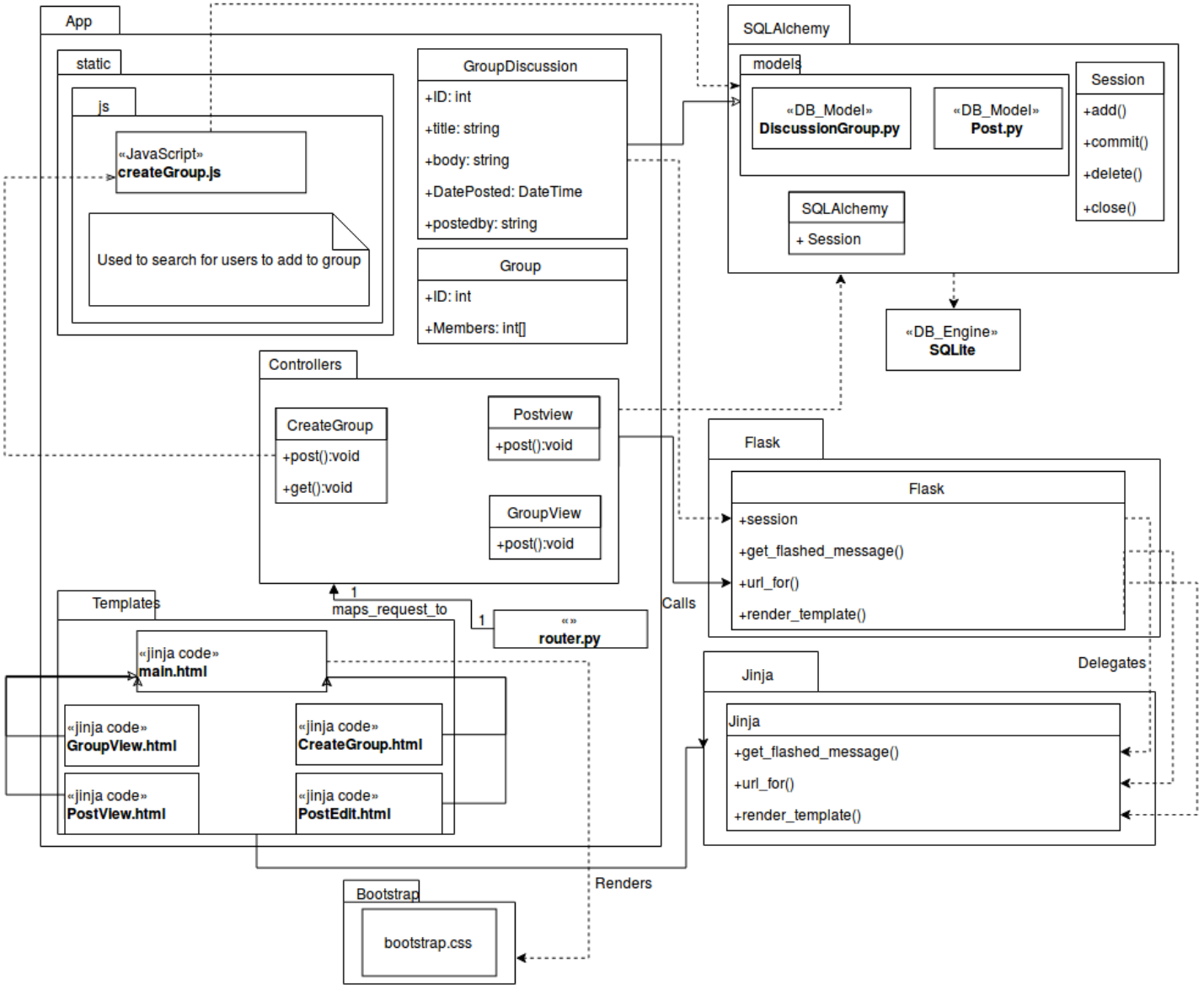
Controllers consists of the python code of the CreateGroup. Those forms are using Flask as a microframework, which is using jinja as a template engine.

The js component, createGroup.js, is used to search for users to add to a group. This is dependent on the DiscussionGroup.py model which is the database for the Discussion Group.

In the Flask code, the url_for() function will build a Url for a specific function. The render_template() function will render a HTML file which base on the jinja and html code. The get_flashed_message() passes a message o the next request, which generally is a template.

The templates folder consists of main.html, GroupView.html, CreateGroup.html, PostEdit.html and PostView.html which are base on jinja and html code. The CSS of the html files will be using the bootstrap library.

Controllers file, routes.py, templates file are all included in app file.


App
static
js
«JavaScript»
createGroup.js
Used to search for users to add to group
GroupDiscussion
+ID: int
+title: string
+body: string
+DatePosted: DateTime
+postedby: string
Group
+ID: int
+Members: int[]
SQLAlchemy
models
«DB_Model»
DiscussionGroup.py
«DB_Model»
Post.py
Session
+add()
+commit()
+delete()
+close()
SQLAlchemy
+ Session
«DB_Engine»
SQLite
Controllers
CreateGroup
+post():void
+get():void
Postview
+post():void
GroupView
+post():void
Flask
Flask
+session
+get_flashed_message()
+url_for()
+render_template()
Calls
Templates
1
maps_request_to
1
«»
router.py
«jinja code»
main.html
«jinja code»
GroupView.html
«jinja code»
CreateGroup.html
«jinja code»
PostView.html
«jinja code»
PostEdit.html
Jinja
Jinja
+get_flashed_message()
+url_for()
+render_template()
Delegates
Renders
Bootstrap
bootstrap.css