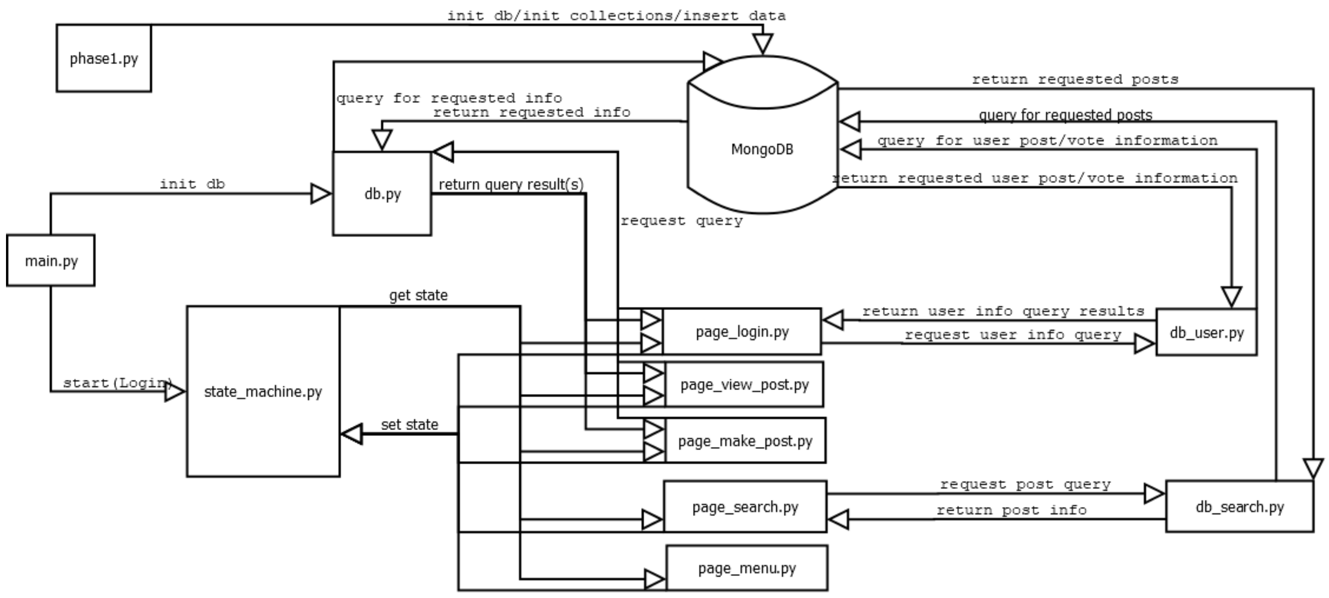
PART A: A general overview of your system with a small user guide

Our system provides a user interface that allows a user to perform actions typical of a question-answer forum. The user can operate as a returning user, new user, or guest. The user can post a question and/or answer, and vote on questions and answers. The user can exit the program if they wish. The below diagram displays the flow of information between files/major components. This flow illustrates how the components are related.



PART B: A detailed design of your software with a focus on the components required to deliver the major function of your application

Technical requirements:
→ python3 (all libraries used are native python: pdb, math, datetime, os, re, json)
→ pymongo
→ mongodb server /client(v 4.4)

Conceptual requirements: Below are the Primary classes and their respective *functions*. The diagram in part A demonstrates how relationships among functions/classes and how they are related.

Phase1:
File Phase1.py: this file initializes a mongodb database and its the collections
→ function main: reads in a user defined port, default if port not provided, and initializes a mongodb
→ all other functions in phase1 initialize a mongodb collection from json files in the CWD.

File Util.py: this file is composed of various utility functions used in Phase1.py

Phase2:
File main.py: this file is the driver that initializes the database and state machine
→ function main: takes in user defined port, initializes a mongodb, initializes & starts state machine

File state_machine.py: this file drives the state machine
→ class State/class StateMachine: runs provided state depending on state (determined by program)

File db.py: this file is primarily composed of utility functions that interface with the database
→ functions guestLogin/register/login: determines if user will be using the system as a guest, new user, or returning user
→ function insertVote/insertTag/insertPost: insert information into the database. The information inserted is the latter half of the name of the function...
→ functions getPost/viewPost: both return a single db post, viewPost additionally increments the viewCount
→ function getAnswers: returns the answers associated with a question being viewed

File page_login.py: this file is composed of the login class and related functions
→ class Login: drives the Login state of the state machine. Verifies user provided login information.
→ function displayUserInfo: utility function to ...display...user...information.

File db_user.py: this file is composed of the function to display new/returning user information
→ function questionsAnswersOwned: gets posts owned by the user and their avg score counts
→ function votesRegistered: gets the votes registered to a user

File db_search.py: this file is composed of functions to get the posts matching the search results of user provided keywords
→ function getMatchingQuestions: gets the matching posts containing keywords in the posts title, body, and/or tag fields
→ function removeDuplicates: functions to remove duplicate posts from the search results

File page_make_post.py: this file is composed of the postQuestion/PostAnswer classes
→ class PostQuestion: drives the PostQuestion state of the state machine. Verifies user provided question information. Afterwards, navigates to a different state based on user provided selection from a menu.
→ class PostAnswer: drives the PostAnswer state of the state machine. Verifies user provided answer information. Afterwards, navigates to a different state based on user provided selection from a menu.

File page_menu.py: this file is composed of the Menu class
→ class Menu: drives the Menu state of the state machine. Afterwards, navigates to a different state based on user provided selection from a menu.

File page_search.py: this file is composed of the Search class and related functions.
→ class Search: drives the Search state of the state machine. Lets the user display the next page of results, search again, view a post. Afterwards, navigates to a different state based on user provided selection from a menu.
→ function inputKeyWords: function to validate provided keywords
→ function displayPage: formats and displays a page of the posts returned by the search from the provided keywords. (5 posts per page)

File page_view_posts.py: this file is composed of the ViewPost class and related function regarding the actions a user can take after viewing a post
→ class ViewPost: drives the ViewPost state of the state machine. Lets the user see full details of a post. The user may perform post actions on this post. Afterwards, navigates to a different state based on user provided selection from a menu.
→ function voteOnPost: allows a user to vote on the post they are viewing.

→ functions showAnswers/displayAnswerPage:allows a user to display answers for the post they are viewing. Allows a user to perform actions on these answers.

File Util.py: this file is composed of various general utility functions

PART C: Your testing strategy

Testing was primarily explorative. It was done on lab machines and on local machines. Test cases involved expected and unexpected input. As a general rule, code was tested before it was pushed to the repository

PART D: Your group work break-down strategy.

Leah (copeland):
→ breakdown: phase1, login/register functions and related utilities, partial db search, search utilities, insert vote / insert tag functions, check spec every 5 seconds, verify correctness of all files and make changes to satisfy spec if necessary, testing, report.
→ time estimate: 8 hours

Abdul-Azeez (Abass):
→ breakdown: phase1, phase1 optimization, state classes and state machine, partial db search, displaying search results and answer result pages and related functions, insert post function, phase 2 utils, verify correctness of all files and make changes to satisfy spec if necessary, testing
→ time estimate: 8 hours

progress made:
We learnt how to integrate a NoSQL database into a TUI console program. We learned about efficiently querying a database via indexing, and how to operate on larger data sets. Furthermore, this project provided the opportunity to practice using version control and working on a team :handshake: .

Coordination:
we used a private github repository for the codebase and discord (messaging, voice call, screen sharing) to communicate and keep the project on track. We were well documented and left “To Do” comments so the other member could understand functionality and pick up where the file was left off.

Assumptions:
→ that if a score on a user question is owned - this 0 impacts the average
→ That a user can post an empty question (Title, body, tags)
→ A user will be logged out upon exiting the program

Limitations:
→ None we are perfect