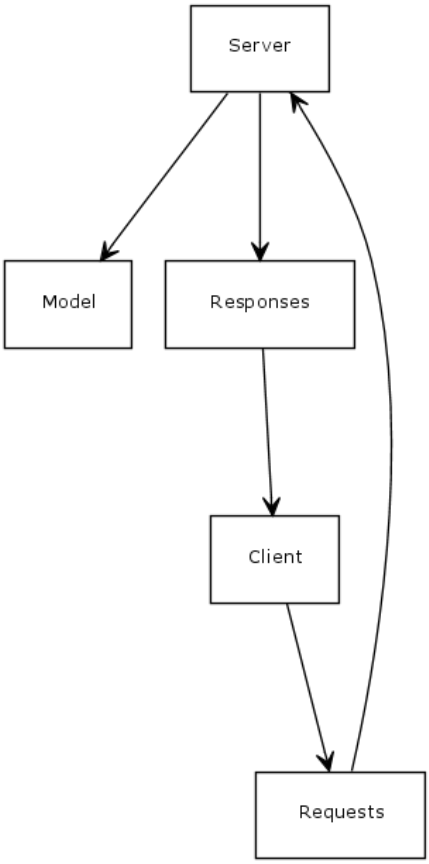
Design Report

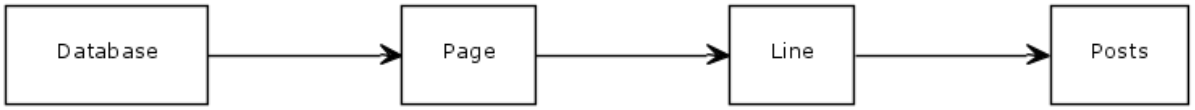
The Assignment has been completed up to Step 3 of the demonstration. Both push and pull functionality.

The architecture of the system was designed in three components:

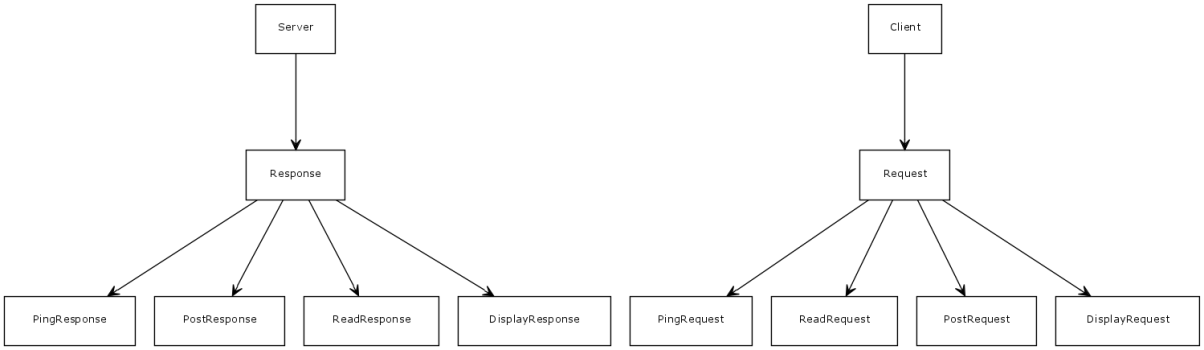
- The underlying model of the Books database
- The networking interfaces, both server and client
- The Requests and Responses objects and their corresponding format and structures



The Model of the system structured around the fact that each line in a page can hold a discussion thread and so The model contains a collection of pages, which contains a collection of lines, which contains a collection of user posts



The networking interfaces contains both the server interface and the client interface, the interfaces users a collection of different requests and responses objects to convey their intentions to each other.

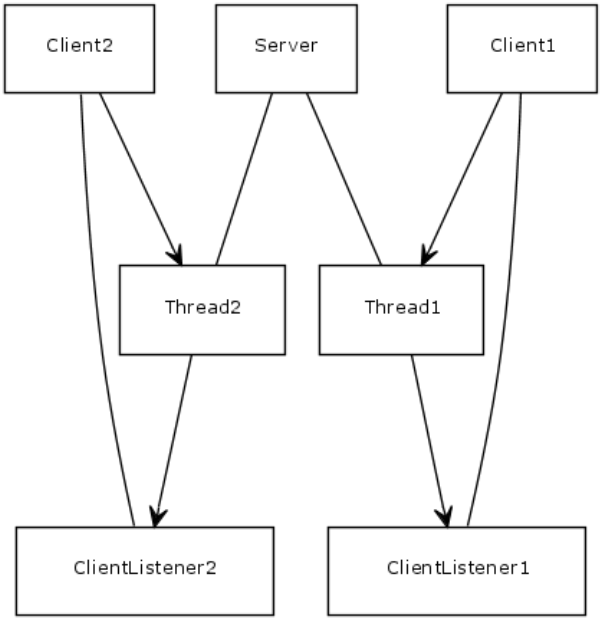


All the Requests classes holds different information that the request needs for its own processing. The data is required is encapsulated within the object and sent across to the server to be processed.

All Requests classes override a common function named process() that produces a Response object with the response information. The processing logic of the requests in not located in the server class, but it is separated away and encapsulated within its own request class. This reduces the amount of coupling between requests and server

The Response classes are similar to the request class as each subclass of Response holds only enough information for its own purposes

All response classes override a common function named toString() which produces a string based output ready for the client to display onto the client's screen. The presentation logic is encapsulated within the response class and so the client code is not burdened with code that is used for printing information, but only for the networking logic.



When the client connects to the server, a new server thread is invoked to handle that particular connection. The thread waits only for that client's input and responds accordingly.

The Client is required to listen to both the user's keyboard input and the server's messages, and so the client runs a parallel thread to listen to all messages arriving from the server. All messages (both keyboard and polling ping messages) are still sent through the main thread of client program and the listener thread receives all the responses and displays them accordingly.

The polling interval ping is achieved by also running a new thread, each ping runs a new thread for the sole purpose of sending the ping request and then shuts down while the listener waits for the ping response.

Due to the possibility that at any time, there could be 3 parallel threads for the client, one listening and two able to send. There is a possibility that the ping request and a keyboard re quest can occur at the same time, causing both messages to be sent down the same stream mixed, creating a corrupt message. This corruption can be solved by using a synchronised write function to ensure that threads never utilise the stream at the same time. However, there is a delay, although extremely tiny, due to one of the messages having to wait. If the keyboard requests are requesting huge amounts of data for processing, the ping delay may become noticeable.

Due to the use of threads on the server side to handle multiple clients, the push functionality required communication between threads about their client's push or pull mode. I solved the thread communication by allowing each thread access to all the push client's tcp connection so that the thread with a post request can forward the post onto the corresponding clients. This level of access is a possible security breach that I have taken in order for simplicity on the server side. A safer approach would be to instead share a list of posts to push instead of sharing a list of tcp connections where each corresponding push thread can read off the list and individually push the new posts onto their respective clients.