Distributed Operating Systems

Bazar.com
REST API web store
firs project

Aisha Marmash 11715167

Walaa Dweikat 11715660

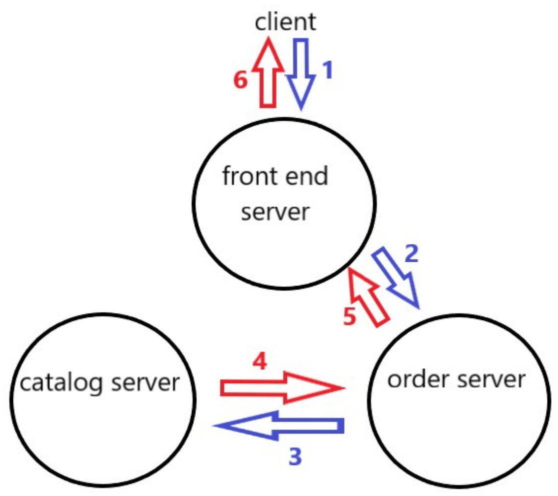
The program is build using multi tiers architecture (2 tiers), it has the front end tier and the back end tier (the server tier). Client tier is one server that is the sends requests to the backed end (server tier). The back end consists of two servers which are the catalog server and the order server. The catalog server is responsible of the database that contains the books in the store and its information, the catalog do update and query request related with the books. The order server maintains the database contains the orders from the clients.
The program is built using flask with sqlite database, it is a web REST API which contains three web
micro-serves .
Every server resides on different machine and they communicate with each other using the network by sending http requests form a server to another.

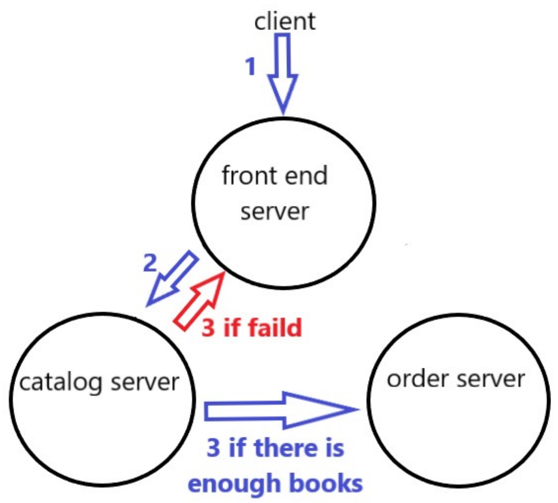
The user how wants to use this program should send requests to the front end tier which will be responsible to communicate with the other two servers to achieve the required functionality. The supported functionalities are:
1 – info(id) : the functionality starts from the user who sends the request http://frontEndServer_IP/bazar/info/<book_id> to the front end server, then the front end server will send the request_
http://catalogServer_IP/bazar/info/<book_id> (this request is will be sent to the catalog server) the catalog server now has a query so the server will search for the book with the book_id and return a json response to the front tier contains the information about the specific book, finally the front tier will send back the json response to the user how did the info functionality.
2- search(topic): what will happen here is the same as the above except of the sent requests that will be different
The request form the user to the front end tier
http://fronEndServer_IP/bazar/search/<topic>
The request from the front tier to the catalog http://catalog_
Server_IP/bazar/search/<topic>
the response is a json which contains information about all of the books which belongs to the same topic.

3- purchase(book_id) : the user will send a purchase request http://fronEnd_IP/purchase/<book_id> to the order server, an optional parameter called amount is send with the request body to specify the book amount to be purchased, it has a default value equals one in case there were no parameter sent with the body request. At first the order server will send a query request to the catalog server to check if there is a sufficient amount of purchased book, the request to be sent is http://catalogServer_IP/bazar/available/<book_id>. If the order server get a response that there is a sufficient amount it will sent an update request http://catalogServer_IP/bazar/decrease_quantity/<book_IP> to the catalog to decrease the purchased book's quantity with by amount and the order store this order inside it's database ,finally, the order will send the following json response to the front end tier {"msg":f"bought book '{temp2.get('book_title')}'"}. If there were no enough amount of books in the store the catalog will send a response telling the order that the amount of books in the database are not enough, the order will also send a json response to the front end telling that there is no enough books.

There is also another functionalities like showing all of the books and it's information, there are the functionalities could be done by the admin of the book store which are showing the list of order , updating any book's price, increasing and decreasing the quantity of any book.

As an improvement to the design, the purchase request could be sent to the catalog server instead of the the order so that if there a sufficient amount of books the catalog will complete the purchase process and send a POST request to the order to add an new order record to its database but if there is no enough amount of books, the catalog will send to the front that the books quantity is not enough. By doing this, the number of sent request from the client to the order and from the order to the catalog will be decreased especially in case of there are no enough amount of books, the response will be sent directly form the catalog to the front tier instead of going though the order server.





Befor

After

The REST API is working on the top if http protocol, so it's security follows http protocol security, in another words, it is may be not secure, to increase the security we could use https protocol or we could add another tier for authentication and encrypt the sent request from the client to the servers.

To test the program we used two virtual machines and the real OS ,we put every server in a machine and the client was in the same machine of the front end. To run the program you could do the same thing and as a client you could use postman to send requests to the front tier or any other command line you like. We used bridge network and static IP to setup the communication between the 3 machines.