Lab 2 Design Doc

We used the Flask library in Python to implement a web framework with REST APIs.

Catalog Server:

Catalog server exposes two operations, query and update, which have corresponding root URLs /query and /update. The arguments that are needed to perform an operation are passed along in URL. For example, to list all books that have topic of distributed systems, a request would be made to /query/distributed_system and to search item 2, a request would be made to /query/2. A json object is returned as the query response. Examples of the json responses are shown below,

```
/query/distributed_system

{"items":{"How to get a good grade in 677 in 20 minutes a day":1,"RPCs for Dummies”:2}}

/query/3

{"Xen and the Art of Surviving Graduate School”:{"COST":150.0,"QUANTITY":13}}
```

Update service works in a similar manner. The service supports three operations on two fields. Three operations are “increase”, “decrease” and “set”. Two fields are “cost” and “quantity. The update service only checks for validity of the operations and fields to update. It doesn’t check if a request would result in an illegal update such as decrement a 0, which could occur when queries and updates are interleaved.

In the cases where quantity or/and cost need to be retrieved or updated, a lock specific to that row of data has to be obtained first. This makes sure the data would be in consistent states and it prevents the aforementioned illegal update from happening.

Order server:

The order server exposes two operations, buy and listing all orders, which have corresponding root URLs /buy and /orders. Listing all orders is a operation only used for experiments and debugging, the frontend is not allowed to list all orders. The argument for conducting a buy is the item number. For example, to buy item 1 a request would be made to /buy/1. A json object is returned as the buy response. Examples of the json responses are shown below,

```
/buy/1

{“catalog_id”: 1,
 “is_successful”: true,
 “order_id”: 3,
```

```
 “processing_time”: 0.01,
“title”: “How to get a good grade in 677 in 20 minutes a day”}.
```

A buy operation calls the query API of the catalog server, checks whether the stock is positive, and calls the update API if the stock is positive. The order server will only call update to decrease the stock by 1, since the lab handout indicates that customers should only be allowed to buy 1 item at a time. All other functionalities of /update are reserved for testing purposes.

The order server is also responsible for tracking whether an order failed or not - if the stock is 0 when queried, the returned json will indicate whether the buy was successful or not.

Frontend server:

Frontend has three operations search(topic), look(item_number) and buy(item_number). Each operation redirects the client request to the responsible server and formats the json response in a readable way before it returns a response that contains string data to the client.

Miscellaneous design choices:
The stock is periodically updated by the order server. If the order server makes a query and realizes the stock of an item is low, it increments the stock by a set amount. However, we allowed this feature to be turned on and off for testing purposes. When we have concurrent client requests, we want to ensure that an illegal decrement never occurs. This can be trivially circumvented by constantly keeping the stock positive with periodic update, so we turned off periodic update for most tests.

Improvements and extensions:

A possible extension is to allow clients to buy more than 1 item at a time from the frontend. Although this is not explicitly required by the handout, it is very sensible from an online store perspective. This would require adding an item quantity parameter to the buy API for the frontend and the order servers. The catalog server already supports multi-item buy for testing and editing the stock.

Another possible improvement is to make the flask server ports more flexible. Right now the entire web framework absolutely requires the order, catalog, and frontend servers to be hosted on specific machines are ports. For greater flexibility, we could make the ports to be taken as command line arguments for the startup script, and pass the ports to python when running the Flask app.