Computer Engineering Department

Distributed and Operating Systems

Design Document

|  |  |
| --- | --- |
| Shima Osamah | 11716299 |
| Shatha Abu Hanesh | 11717771 |

* **Front server**

Three main services are implemented:

1. Search:

* We implemented search ether by (https://front\_IP/search/<category\_name>) which will return all books id, and titles that are within this category.
* OR by just (https://front\_IP/search) which will return all books id, and titles.
* Search then will do a GET request to Catalog server about that category or all book. (requests.get(http://catalog\_ip/query/search/<category\_name>)) or without (<category name>)
* The output will be returned in JSON format.

1. Info:

* We implemented info ether by (https://front\_IP/info/<book\_id>) which will return the book title, price, quantity.
* OR, by just (https://front\_IP/info) which will return all books title, price, quantity.
* Info then will call GET request to Catalog server with book id or without,

requests.get (http://Catalog \_IP/query/info/<book\_id>).

* The output will be returned in JSON format.

1. Purchase:

* Purchase API could be used like this (https://front\_IP/purchase/<book\_id>), and book id is mandatory or an error massage will be return.
* Purchase then would do PUT request to Order server to buy that book.

requests.put(http://Order\_IP/purchase/<book\_id>).

* A massage (buy book <book\_name>) would be returned if the book is existing and its quantity is not zero. Or an error massage will appear.
* **Replication**
* Two main services are implemented in both replicas (order and catalog):

1. Query:

* Query could be for category (http://Catalog\_IP/query/search/<category\_name>),

or just (http://Catalog \_IP/query/search). JSON with all books id and title would be returned.

* Query could be for book info (http://Catalog\_IP/query/info/<book\_id>),

or just (http://Catalog \_IP/query/info). JSON with all books title, price, and quantity would be returned.

1. Update:

* Update service will chick if the book id is available and it’s not all soled, where error massage would appear, and then it will update the book quantity number by decrement it by one, and success massage will be returned.
* Consistency: (Primary-based: remote write)

Catalog server is the

In case of update, this request will be handled in one of the replicas, in case it was in catalog, after it calculate the new value it will send alert to order server if it replays back with done response then the update will complete and send back success massage.

If that request reaches order server first, the it will forward the request to catalog server and when it’s done it will response back to order server with the update, and the last will reply with the result.

* **Caching**

All search and info requests are cached in the frontend server, it’s capacity is limited to 10, whenever its full the oldest one, which would be in the front of the array, would be deleted, in case an update request is done, all related request to that item will deleted from cache.

* **Technology**
* Flask –python: to implement our servers
* CSV files: to represent the database
* Dictionary data type for cache
* VMware to run the servers on Ubuntu VM.