Code offers:

1. Two gateways to call the REST APIs.
2. Data replication using leaderless replication (with two copies).
3. Consistency through read-repair.
4. Partition using CRUSH hash function.
5. Mapping of database to backend application suing Zookeeper.
6. Offers ADD/UPDATE/DELETE/LIST User cart to user
7. Offers LIST ALL USER CART and LIST ALL ITEMS IN ALL CARTS to admin
8. Concurrent writes are possible and its resolved in backend properly.

ASSUMPTION:

1. USER MUST CALL /listusercart before any update or delete command, otherwise he cannot update/delete.
2. Using .json files for data storage
3. There are 3 Nodes (1 main and two replicas as folder named n1 n2 n3 in /data/main_nodes) and each node has 3 partitions to divide data (p1.json, p2.json, p3.json as 3 files in each n1 n2 n3 folder.
4. There are 3 nodes for secondary index, as files named p1_sec.json, p2_sec.json, p2_sec.json inside /data/secondary_nodes/ path.
5. Read Quorum = 2 and Write Quorum = 2 {CAN BE CHANGED INSIDE CODE}

Table Schema:

1. Main Data (primary index = userId)
   Data is stored in p{number}.json file as dictionary of dictionary with first key as userID and Keys of userID{} as item = count and last_updat_time = val.
   ```
   {
     "userId": {
       "Item1": count,
       "Item2": count,
       "|last_update_time|": "2020-11-29 23:15:09.153365"
     }
   }
   ```
2. Secondary index = item. Again stored in .json file as:
   ```
   {
     "item1": {
       "Count": val,
       "User List": [
         "userId1",
        “userId2”
       ]
     }
   }
   ```

How to Run:

1. Install Docker/docker-engine/docker-compose/docker-network
2. Go to **/Code/Main** file and run the command "docker-compose build"
3. Docker image will build, then run "docker-compose up"
4. 4 containers of Docker will run:
   a. Gateway 1 mapped to localhost:5000 of local machine
   b. Gateway 2 mapped to localhost:5002 of local machine
   c. Zookeeper connected to both gateway 1 and 2.
   d. Common volume named "main_shoppingcart_data" which contain json files to store data. This volume is assessable to both gateways.
5. Open UI1.html and UI2.html, these are UI for making REST queries to above both gateways.
6. Manipulate data using UI.

Design:

User 1 enters a data {userId: "1"; item: "mobile"; count:10} + {userId: "1"; item: "laptop"; count:5}.

a. UserId = 1 is mapped to partition to p1 according to crush map.
b. Now Write Quorum = 2, hence randomly two nodes are for writing, let say n1 and n2. So data file names and its path are taken from mapping of Zookeeper.
c. First the data is written in n1/p1_temp.json AND n2/p1_temp.json file (with time stamp) as:

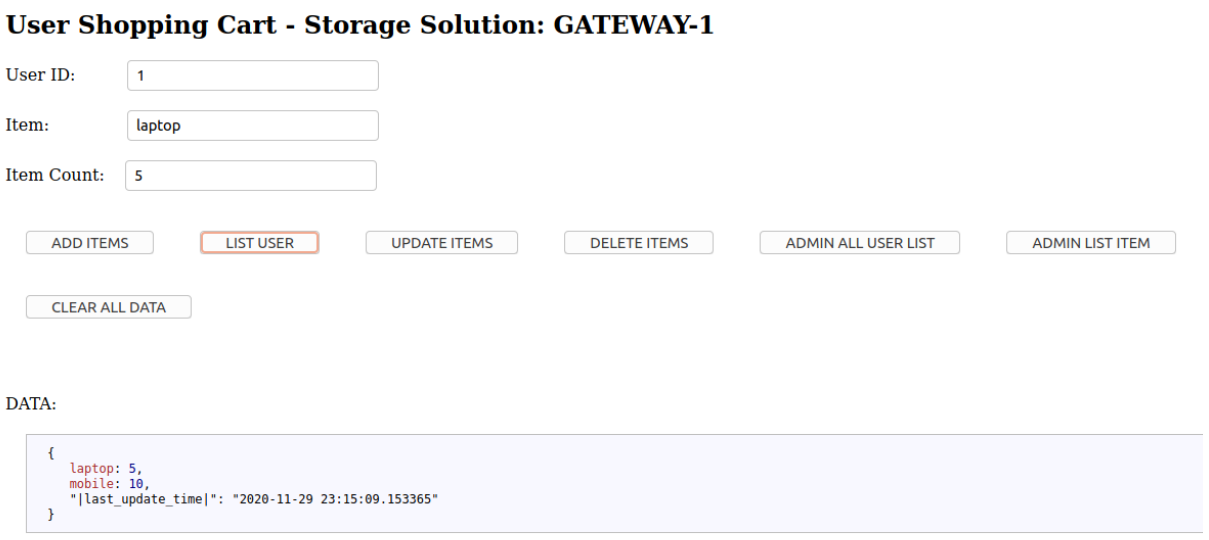
```
{
  "1": {
    "2020-11-29 23:13:56.521098": [
      "mobile",
      10,
      "add"
    ],
    "2020-11-29 23:14:05.952398": [
      "laptop",
      5,
      "add"
    ]
  }
}
```

   Due to time stamp, all the concurrent writes are resolved.

d. Now when user list his shopping cart, Read Quorum = 2, so randomly two nodes are selected let say n2 and n3. Now n2/p1_temp.json file has above data but n3/p1_temp.json does not have. So all the data from both files is merged. And for each time stamp in ordering, this transaction data from _temp.json file is deleted and is written to main file (according to operation) with new time stamp: n2/p1.json AND n3/p1.json as: (but main file.. n1/p1.json is still empty, or contains stale data, which will be updated when user calls this node)

```
{
  "1": {
    "laptop": 5,
    "mobile": 10,
    "|last_update_time|": "2020-11-29 23:15:09.153365"
  }
}
```

DATA IS SHOWN IN UI:



User Shopping Cart - Storage Solution: GATEWAY-1

User ID:	1
Item:	laptop
Item Count:	5

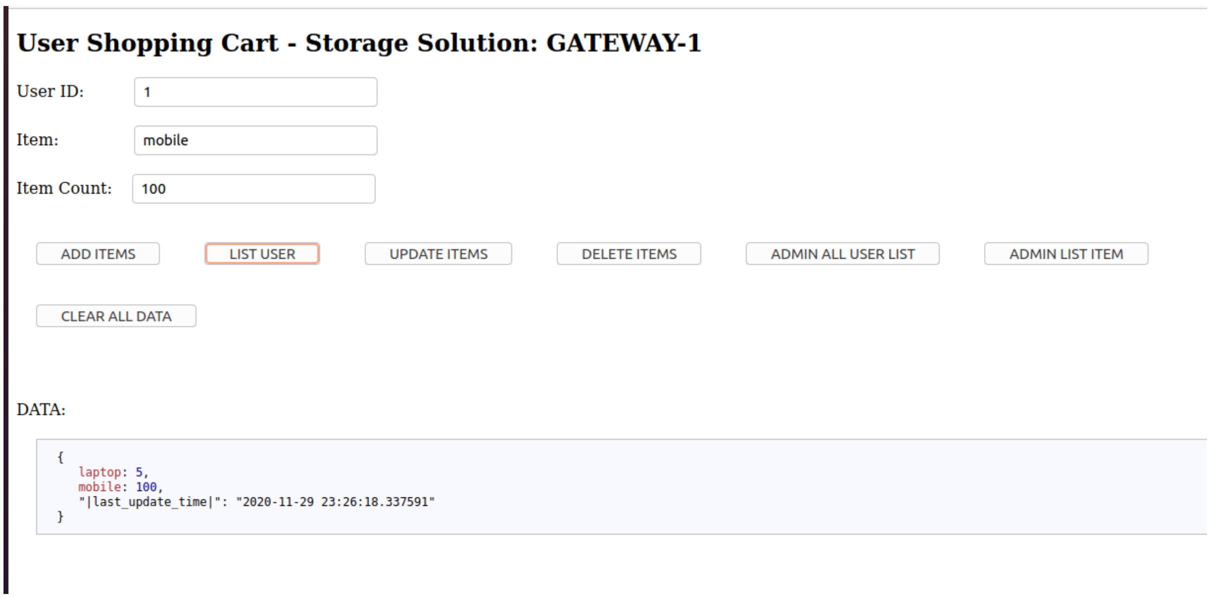
DATA:

```
{
  laptop: 5,
  mobile: 10,
  "|last_update_time|": "2020-11-29 23:15:09.153365"
}
```

e. When update is called as {userId: “1”; item: “mobile”; count:100}. It is again stored in _temp.json file with any 2 nodes selected (as read quorum = 2)

```
{
  "1": {
    "2020-11-29 23:25:43.323877": [
      "mobile",
      100,
      "update"
    ]
  }
}
```

And when user list item. It is shown to him:



User Shopping Cart - Storage Solution: GATEWAY-1

User ID: 1

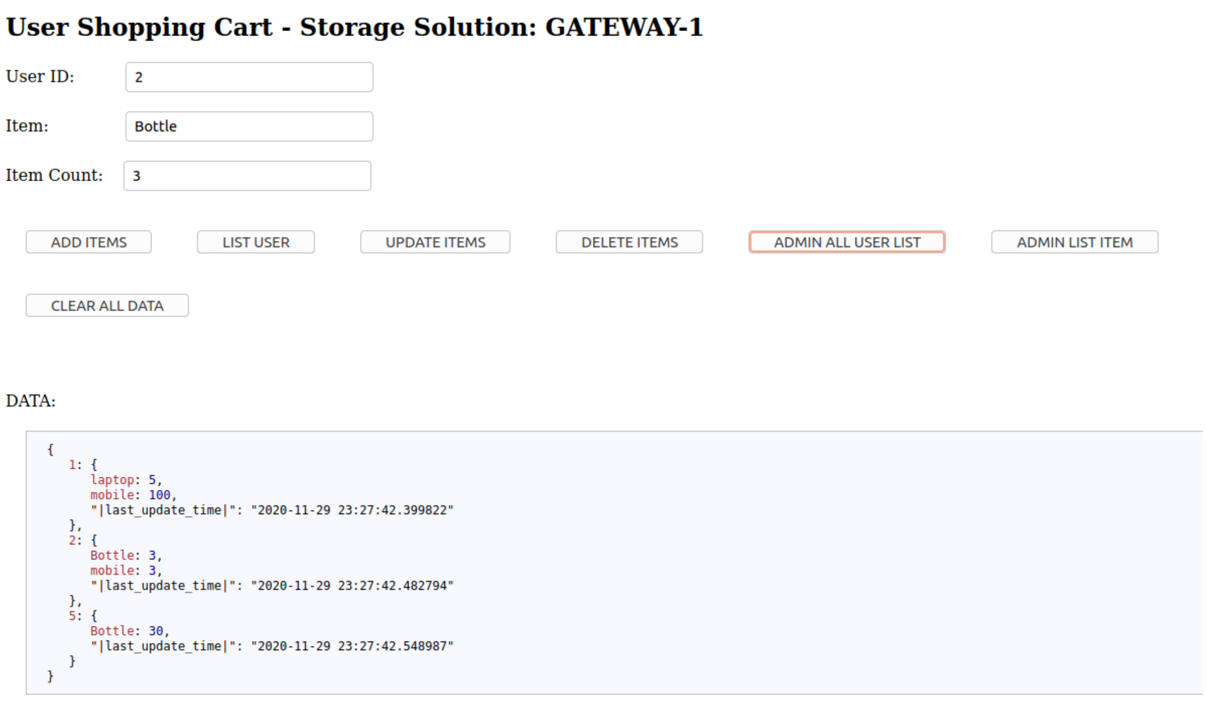
Item: mobile

Item Count: 100

DATA:

```
{
   laptop: 5,
   mobile: 100,
   "|last_update_time|": "2020-11-29 23:26:18.337591"
}
```

f. When admin list all the user_cart, It is shown as:



User Shopping Cart - Storage Solution: GATEWAY-1

User ID: 2

Item: Bottle

Item Count: 3

DATA:

```
{
   1: {
      laptop: 5,
      mobile: 100,
      "|last_update_time|": "2020-11-29 23:27:42.399822"
   },
   2: {
      Bottle: 3,
      mobile: 3,
      "|last_update_time|": "2020-11-29 23:27:42.482794"
   },
   5: {
      Bottle: 30,
      "|last_update_time|": "2020-11-29 23:27:42.548987"
   }
}
```

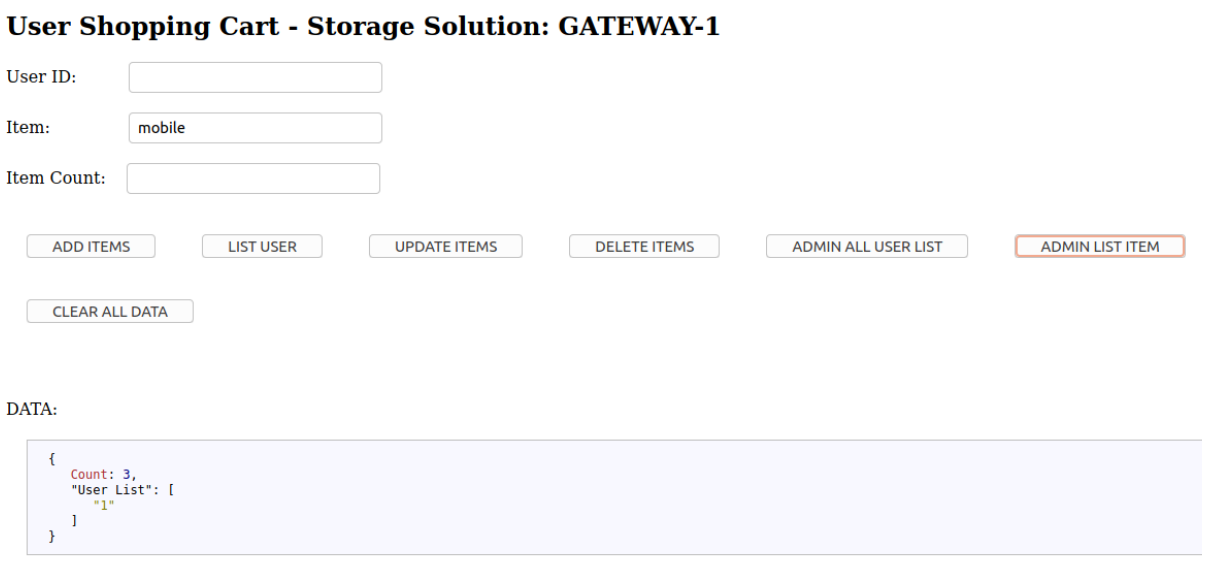
Secondar Index:

1. There are 3 nodes for storing secondary index, and secondary index = item.
2. This index it term partitioned and the partition number of index is calculated by crush map using sum of bytes of item string.
3. When user enters any item, let say “mobile”. According to crush map it is stored in p1 as dictionary with one key as count and another key as list of users having it in their cart:

```
{
   "mobile": {
```

```
            "Count": 3,
            "User List": [
              "1"
            ]
          }
        }
```

4. And when admin list this item “mobile” it is shown in UI as json format:



User ID:

Item: mobile

Item Count:

```
{
   Count: 3,
   "User List": [
      "1"
   ]
}
```