**KIT514 Assignment2 Documentation**

**Yi Wang**

**546705**

The architecture of my web services consists of 5 sections hosted in two virtual machines:

VM1:

* Bank
* Market
* Seller1
* Seller2

VM2:

* Seller2

Each section contains at least one web service, and they use different databases, meaning they can only visit each other’s resources through their web services:

Bank:

* card\_check.php: take the user ID, card number and card PIN as a POST request to authenticate. If the authentication fails, return a ‘403 Forbidden’ response code. Otherwise, encrypt the user ID and expiration time for 5 minutes as payload with a 200 response.

Market:

* search.php: stores a list of sellers API for visiting and taking an item name as a search query to proceed with string matching. If no item is found, return a ‘404 Not Found’ response code. Otherwise, respond to the seller IP, item IP, item name and price with a success message and a response code 200. The success message will also indicate if multiple items are found in a separate list.
* validate\_card.php: process the validation of JWT issued by card\_check.php. It will take the JWT from the HTTP ‘authorize’ header. If the header does not include ‘authorize’ header or the JWT does not exist in the ‘authorize’ header, it will return a ‘400 Bad Request’. If the token fails to decode, invalid or expired, it will return a ‘401 Unauthorized’ response. Otherwise, it will return the user ID stored in the JWT.
* purchase.php: proceed with purchasing an item for a user if the user is validated. It inputs the user ID, item ID, quantity, and seller IP. Firstly, it will take the JWT from the ‘authorized’ header and use validate\_card.php for validation. After the validation, it fetches the user ID from the token and compares it with the user ID from the POST request. If they do not match, return a ‘403 Forbidden’ response to indicate an invalid credential. Then, it will start to fetch the full details of a single item by requesting item.php with the seller IP and item ID (My customised web server).

Then, it will start to read the item data from the response. If it fails to read the JSON object from the response, return a ‘400 Bad Request’. Otherwise, the item's price and stock will be read from the response. If the quantity the user requires from the POST request is greater than the item’s stock or the user’s current balance is less than the product's price, it will return a ‘400 Bad Request’ indicating insufficient stock or balance. After the check before the purchase, it will proceed with the query. It will deduct the user’s balance, create a new record in the Purchase table in their databases and respond to the details of this newly created purchase object with a 200-status code as a receipt.

* purchase\_search.php: taking either the purchase ID or user ID as GET requests. If none of the requests has been provided, return a ‘400 Bad Request’ response. Then, it will fetch the required purchase record from the database and return it in JSON with status code 200.
* purchase\_cancel.php: taking a purchase ID as a DELETE request. Return a ‘400 Bad Request’ response code if it is not provided. Then, it will check if the purchase exists. If it does not exist, return a ‘404 Not Found’ response. Otherwise, it will refund the user’s balance, delete the purchase record from the database and return the deleted purchase record in JSON with a 200-success response.
* add\_balance.php: taking an amount and user ID as POST requests. It also requires a JWT validation to proceed. Return a ‘403 Forbidden’ response if the user ID in the token does not match the user ID in the POST request. If the charge amount is less than 0, return a ‘400 Bad Request’. If the amount equals 0, then it will try to fetch the user’s information. If the user’s information is not found in the database, it will return a ‘404 Not Found’ response. Otherwise, the user’s balance data will be presented in JSON directly. If the amount exceeds 0, it will update the user’s balance and return the updated user’s balance information in JSON as a receipt.

Seller1 (VM1)/Seller2(VM1)/Seller2(VM2):

* item.php: return the complete information of an item in JSON, including the quantity. (My customised web server)
* itemlist.php: return a list of a seller's items in XML, excluding the quantity information.