*e\_Wallet* v4.0.0

Connection:

1. Set db details on private fields.
2. Set table name carefully.
3. Constructor parameter

* ‘wallet’ for eWallet functions;
* ‘bank’ for bank functions;

1. If parameter is not appropriate, link will be null;
2. If all correct, giveLink will return a proper link to connect.
3. LINK MUST BE CLOSED AFTER EVERY USE.

User list:

1. Give ‘wallet’ as connect to.
2. Else link will be null, ‘Connection problem on server’ will be send as json response. With error code 404.
3. After that request will be checked(validate).
4. Data is null, means not payload. If false, simply return ‘Get lost’;
5. If valid request.
   1. Decrypt request data using nirapotta, there are a tool class.
   2. Key is ‘giveIds ‘; with a space in the end.
   3. isBeg will return true if valid.
6. Fetch data from DB and send list as json response. Code 200;
7. If isBeg invalid, it will return ‘thiknai’ as json response. With error code 404.

Purchase:

1. Give ‘wallet’ as connect to.
2. Else link will be null, ‘Connection problem on server’ will be send as json response. With error code 404.
3. Then check validity of request. If invalid response with “Get lost”.
4. Store data to purData class, then decode.
5. Check algorithm;

* Amount;toID
* PIN
* Or
* Amount;Pin
* toID

1. “Data, sec, body, ano” accordingly will be the request. In encoded format.
2. If sec len is 4 then it must be algo 1. Then extract all data from appropriate field.
3. If valid format found, then start main job.
4. Check bank key.
5. Check user pin.
6. If invalid, send json, “error” as status.
7. If valid, then check for pin validity.
8. If invalid, send json, “invalid” as status.
9. If valid, ready data formate for bank api.
10. “Key, clientFrom,clientTo,am”. Send it.
11. Send it using purData class. Which contains “purchase\_req”.
12. It is for making post request to BANK api.
13. Response will be trid.
14. If trid valid, then store it on rds DB.
15. If any problem occurs(count == 4) then sent for manual input on DB.
16. Send response with status “manual” and trid.
17. If no problem them send status “ok” and trid.

Log in:

1. Check validity of request. (injection).
2. Check credential also app key. On req\_data table.
3. Request format:(json)

* "id": "",
* "pass": "",
* "key": ""

1. Response:

* “status” : “ok”,
* “name” : “”,
* “add”: “ ”,
* “qstn” : “ ”
* “flag” : “”

1. If wrong pass, “status”: “invalid”.
2. If unauthorized access, “status”: “get lost”.
3. If qstn in null on response means eWallet is not registered.
4. If flag bit 1, means transaction open.
5. If flag bit 0, transaction closed.
6. If flag bit null, means eWallet is not registered yet.

Createacc:

1. Check validity of request. (injection).
2. Check credential also app key. On req\_data table.
3. Request format:(json)

* "key" : "",
* "pin" : "",
* "ans" : "",
* "qstn" : "",
* "id" : "",
* “pass”:””

1. “pin” must be 4digit long.
2. If valid request it will make a create request to Bank API(POST).
3. If created then make entry to RDS DB.
4. Response format:

* “status” : “ok”

1. If bank is unable to create AC at that time. “status” : ”connection problem, please try again later”.
2. If pin size invalid “status” : ”invalid pin size”.
3. Any other case “get lost”.
4. If injection detected then “get lost, you fool.

switchTrans:

1. Check validity of request. (injection).
2. Check credential also app key. On req\_data table.
3. Request format:(json)

* "key" : "",
* "pin" : "",
* "ans" : "",
* "id" : ""
* "curr" : "",
* “pass”:””

1. If valid request check pin and ans, then check curr(current status).
2. If curr is 1 update wallet table ‘onOrOf’ to ZERO, if curr 0 then turns ‘onOrOf’ to ONE.
3. Response: (json)

* “status”:”ok”

1. In case of invalid pin or ans(security question answer), “status” : “invalid”.
2. Any other case “get lost”.
3. If injection detected then “get lost, you fool.

changePin:

1. Check validity of request. (injection).
2. Check credential also app key. On req\_data table.
3. Request format:(json)

* "key" : "",
* "pin" : "",
* "ans" : "",
* "id" : "",
* "new" : "",
* “pass”:””

1. If valid request check old pin and ans.(hash)
2. If true then make new pin hash, and update
3. Response: (json)

* “status”:”ok”

1. In case of invalid pin or ans(security question answer), “status” : “invalid”.
2. Any other case “get lost”.
3. If injection detected then “get lost, you fool.

balance:

1. Check validity of request. (injection).
2. Check credential also app key. On req\_data table.
3. Request format:(json)

* "key" : "",
* "id" : "",
* “pass”:””

1. If valid request check pass.(hash)
2. If true then make request to bank “balance” api.
3. Take response then forward it as response
4. Response: (json)

* “status” : “ok”
* “balance” : ””

1. Invalid pass

* “status” : “invalid”

1. Any other case “get lost”.
2. If injection detected then “get lost, you fool.

viewAllHistory:

1. Check validity of request. (injection).
2. Check credential also app key. On req\_data table.
3. Request format:(json)

* "key" : "",
* "id" : "",
* “pass”:””

1. If valid request check pass.(hash)
2. If true then make request to bank “history” api.
3. Take response then forward it as response
4. Response: (print\_r)

* A three dimensional array

1. Any other case “get lost”.
2. If injection detected then “get lost, you fool.

singleHistory:

1. Check validity of request. (injection).
2. Check credential also app key. On req\_data table.
3. Request format:(json)

* "key" : "",
* "id" : "",
* “pass”:””,
* “trid” : “”

1. If valid request check pass, key.(hash)
2. If true then make query to e\_Wallet DB.
3. Take response featch it then forward it as response
4. Response: (json)

* “ststus” : “ok”,
* “data” :” “

1. If problem on DB/ API:

* “status” : “not ok”.

1. Any other case “get lost”.
2. If injection detected then “get lost, you fool.

*Bank*

create:

1. Check validity of request. (injection).
2. Check credential also app key. On req\_data table.
3. Request format:(json)

* "key" : "",
* "id" : "",
* “name”:””

1. If valid request, start transaction.
2. Make entry to user\_map.
3. Make entry to tr\_his with all field zero but, time, id and trid with appropriate value.
4. This is to track that initial balance is zero.
5. Response: (echo)

* “success“

1. If problem on DB/ API:(echo)

* “try again later”

1. Any other case “get lost”.
2. If injection detected then “get lost, you fool.

load:

1. Check validity of request. (injection).
2. Check credential also app key. On req\_data table.
3. Request format:(json)

* "key" : "",
* "id" : "",
* “am”:””

1. If valid request, start transaction.
2. Get current balance and client id.
3. Based on that make new balance and make a new entry on tr\_his.
4. For this case “transaction description” will be ‘Balance credited’.
5. Response: (echo)

* “balance (am) successfully loaded to account“.

1. If problem on DB/ API:(echo)

* “sorry”

1. Any other case “just wait local police will approach you any time”.

balance:

1. Check validity of request. (injection).
2. Check credential also app key. On req\_data table.
3. Request format:(json)

* "key" : "",
* "id" : ""

1. If valid request get current balance and make response.
2. Response: (echo)

* “balance“.

1. If problem on DB/ API:(echo)

* “sorry try again later”

1. Any other case “just wait local police will approach you any time”.

history:

1. Check validity of request. (injection).
2. Check credential also app key. On req\_data table.
3. Request format:(json)

* "key" : "",
* "id" : ""

1. If valid request get all transaction history.
2. Then make response.
3. Response: (print\_r)

* “3D array“.

1. If problem on DB/ API:(echo)

* “sorry try again later”

1. Any other case “just wait local police will approach you any time”.

transaction:

1. Check validity of request. (injection).
2. Check credential also app key. On req\_data table.
3. Request format:(json)

* "key" : "",
* "clientTo" : "",
* “clientFrom” :””,
* “am”:””

1. If valid request, start transaction.
2. Get current balance and client id for both.
3. Generate new balance for both. With deducted rate(bank charge) from both.
4. Check “clientFrom” has sufficient balance or not.
5. For “clientFrom” debit is “am”, credit zero and amount is newly calculated one.
6. For “clientTo” debit is zero, credit “am” and amount is newly calculated one.
7. Finally make transaction.
8. Response: (echo)

* “trid for clientTo“.

1. If insufficient balance:(echo)

* “”insufficient balance”

1. If problem on DB/ API:(echo)

* “try again later”

1. Any other case “just wait local police will approach you any time”.