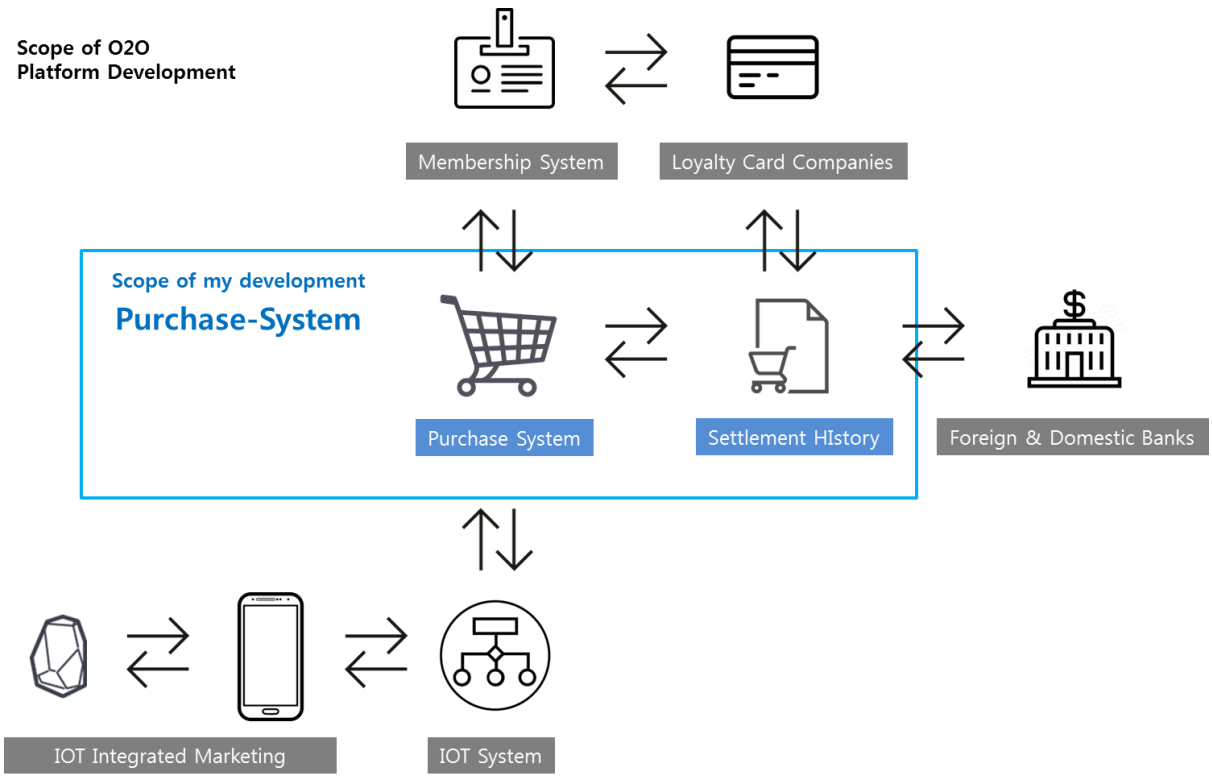
Project of O2O Platform

2016.02.10 ~2016.07.10

Team members (15 people)	
Project Manager (1 people)	Manage Project
DBA (1 people)	System Architecture and Database Modeling
Software Architecture (3 people)	User Interface Architecture
Server Developers (3 people)	Develop Membership, Purchase, IOT System
Technical Assistant (2 people)	Stress test and Server Maintenance and Distribution
Front-End Developers (2 people)	Develop Components for Client webpage
Mobile Developers (3 people)	Develop Android and Mobile Application

Develop Platform	
Develop time	2016.02.10 ~2016.07.10
Operation System	Window8 K 64bit, Mac OX 10.7 Lion, Unix 4.3.20
Programming Languages	Java JDK 1.7.0_45, Objective-C 2.0
Develop Tools	Eclipse 4.42 Lunar, XCode 4.1, Putty 0.64
DB	Mysql 5.7.7, Oracle 12.1.0.2, MongoDB 3.0.2

Scope of O2O Platform Development

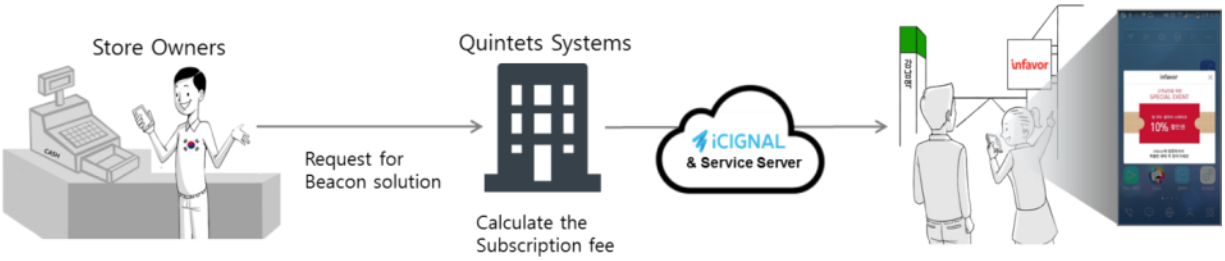


The Problem

Store owners in Korea originally purchased the Quintet System's iCignal, the beacon solution used for marketing purposes by phone.
However the service is now available to customers all around the world including Vietnam, Taiwan and Indonesia. Since the pricing and purchase procedures vary from country to country, a unified system for handling purchases from various countries is essential.

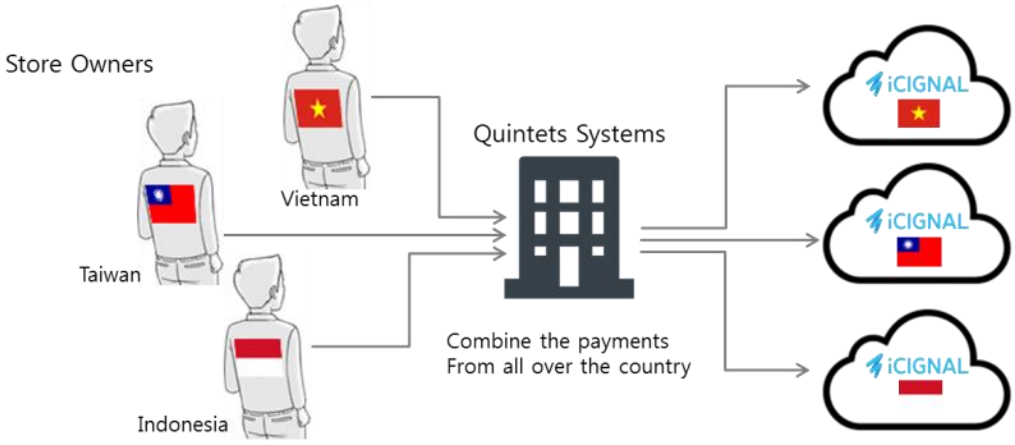
Before the purchase system

The services were provided based on phone-call requests from store owners. However, because customers from various countries asked for Quintet's marketing solution, the purchase system needed to be part of the solution in order to organize the services for each country.



After developing the purchase system

The purchase system helped organize the service for customers from each country and store the purchase history list for more optimized transaction.

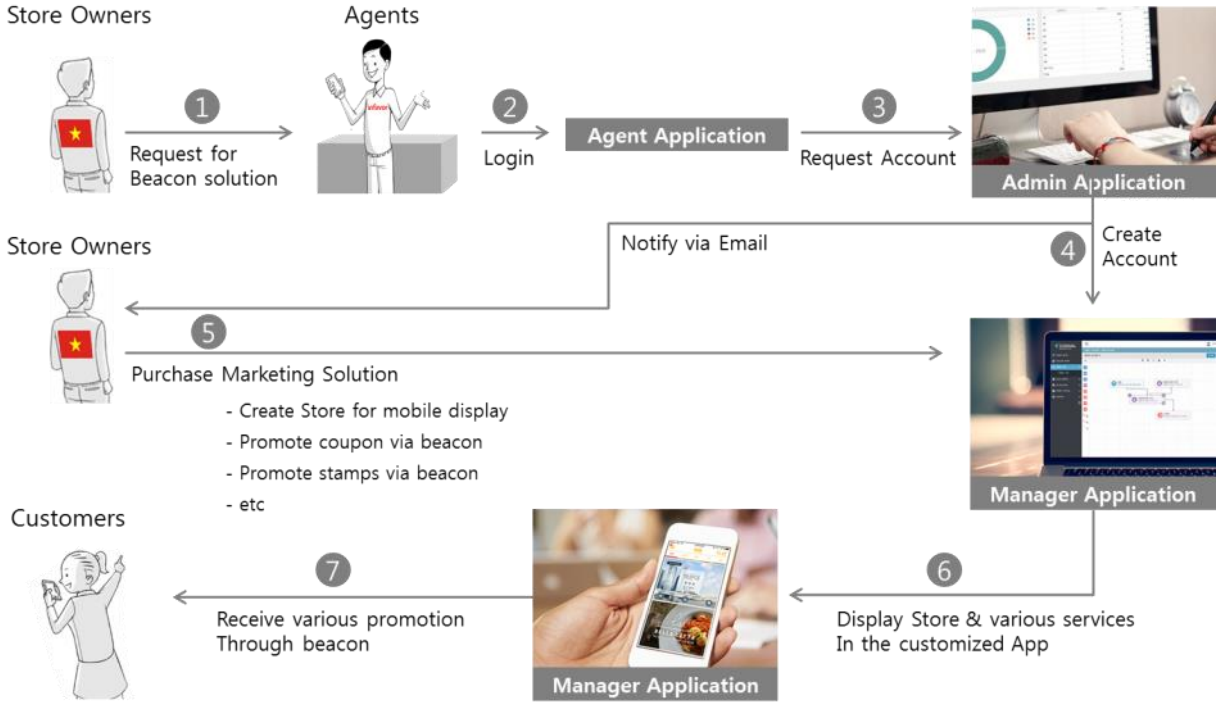


For additional details about iCignal, check the link below

http://quintet.co.kr/product/icignal

The Objective

The agents should receive the store owner's request and make a unified application form and send it to Quintet-Systems Admin Website. An employee from Quintet will decide whether the applicant is acceptable or not on the Admin webpage. When the applicant is proven to be acceptable, the admin will accept the application and this will automatically produce an account for the store owners to use the Manager Website. The store owners may purchase services on the Manager Website in order to promote his or her stores.



Further Requirements

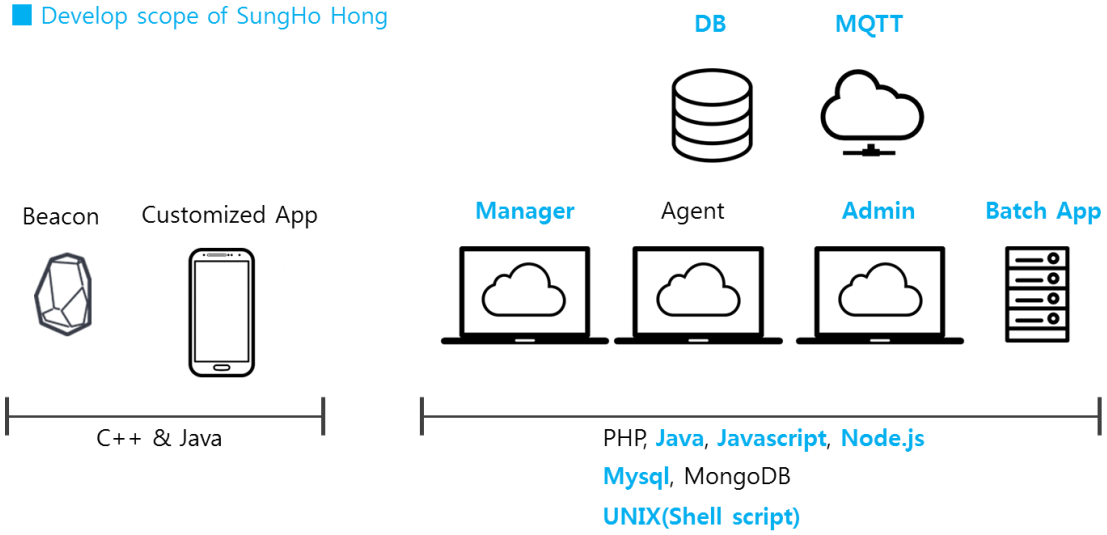
1. Manager Web-Application
   A. The users may purchase and extend their purchase of Quintet's Solution.
   B. The Users may switch their types of service (Quintet's Solution) from premium to economy and the changes should reflect the cost of daily and monthly payment.
   C. Options for post-payment + pre-payment are available when purchasing Quintet's Solution.
   D. Price-Tag data for calculating the costs during purchase should only be retrieved by the database once (the time when the application is started)

2. Admin Web-Application
    A. The pricing and banned texts are different for each country's web applications.
    B. Each pricing data and banned texts should only be retrieved by the database once (the time when the application is started)
3. Batch Application
    A. Consistently check for compromised data and expired contracts
    B. Perform daily and monthly calculations of profit

Scope of my Development Role

I worked as a server side developer and developed all the areas that relates to the purchase system. These areas included database queries supervised by the DBA JungBeong Hee, the MQTT server in cooperation with the technical engineer JinSung Kim, and the batch and web applications.

- Develop Framework components for purchase system
- Develop purchase-related services in Admin Web-Application
- Develop purchase-related services in Manager Web-Application
- Develop purchase-related object files in Batch application



Main Task	Front & Back-End Service Development
	Back-End Framework Component Development
	Development of MYSQL Procedures and Functions
	Shell Scripting
Co-op Task	Query Tuning with DBA, JungBeong Hee
	Stress test of MQTT server with TA JinSung Kim

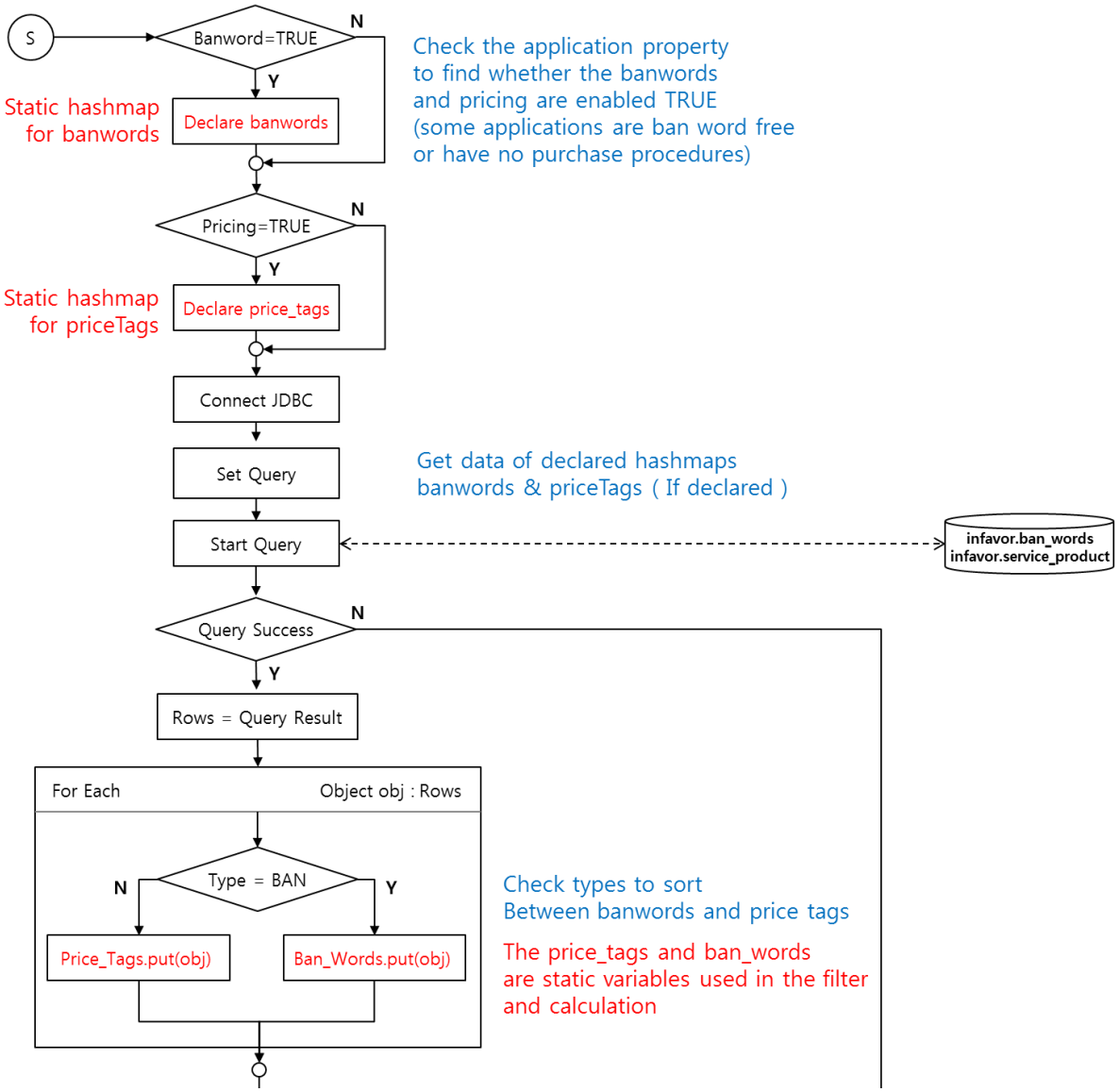
Framework Components

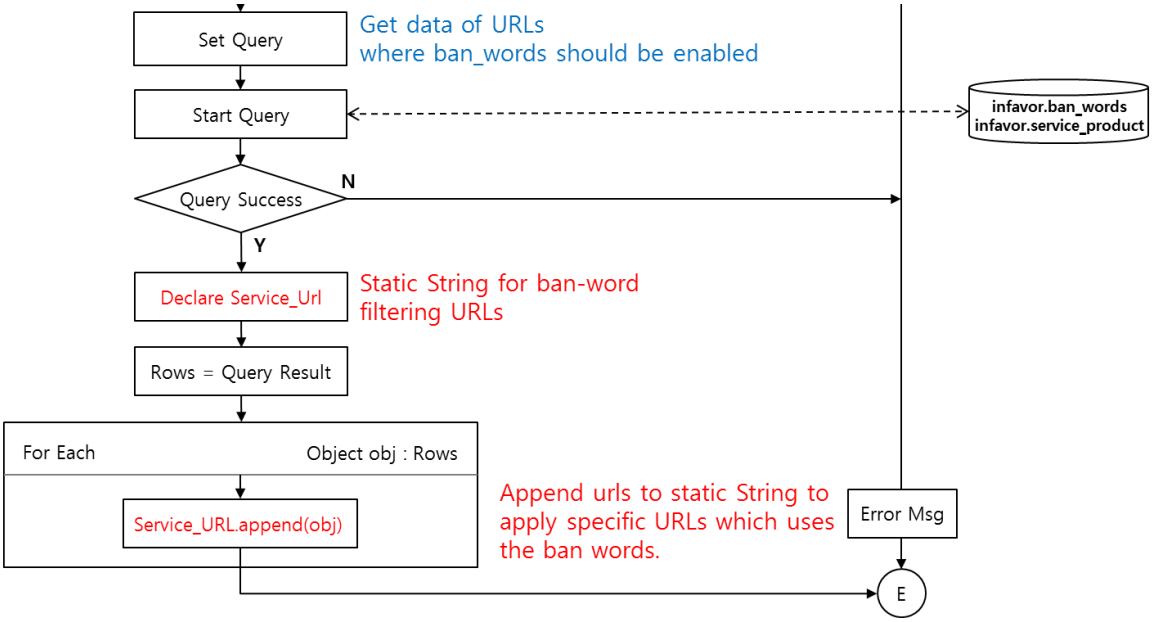
Description

The filtering program for ban words and the MQTT connection are developed inside the common framework of the application.The framework is mostly composed of singletons and static data. The framework initializes when the application starts and is valid until the application is terminated. The framework is applied to both the Admin and Manager websites of various countries.

01.Initialize the static data

① Receive the pricing & banwords data from the database only once
② Receive URLs that filter banwords from the database only once
③ Map the pricing, banwords and URLs in the static variables of the framework



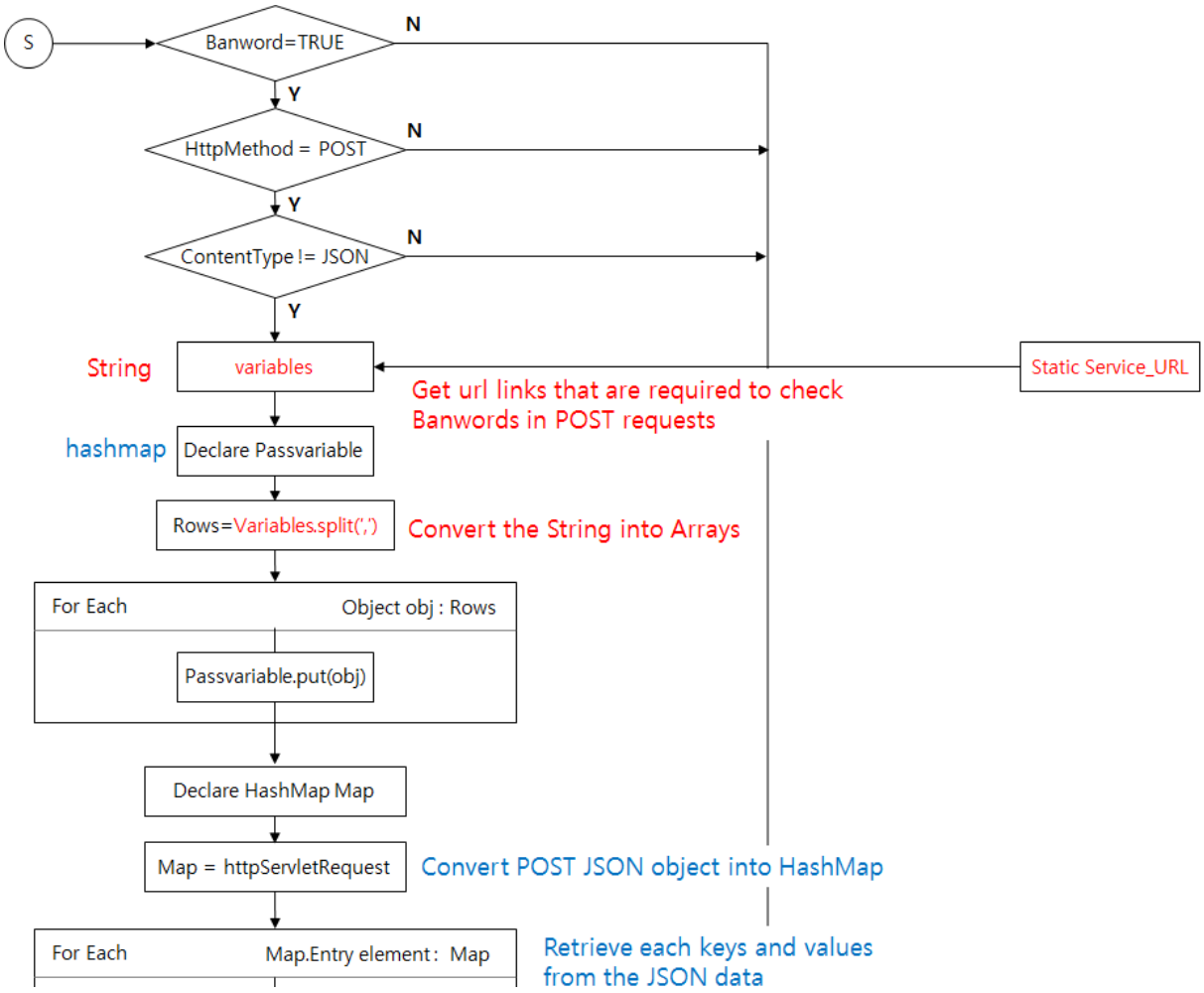


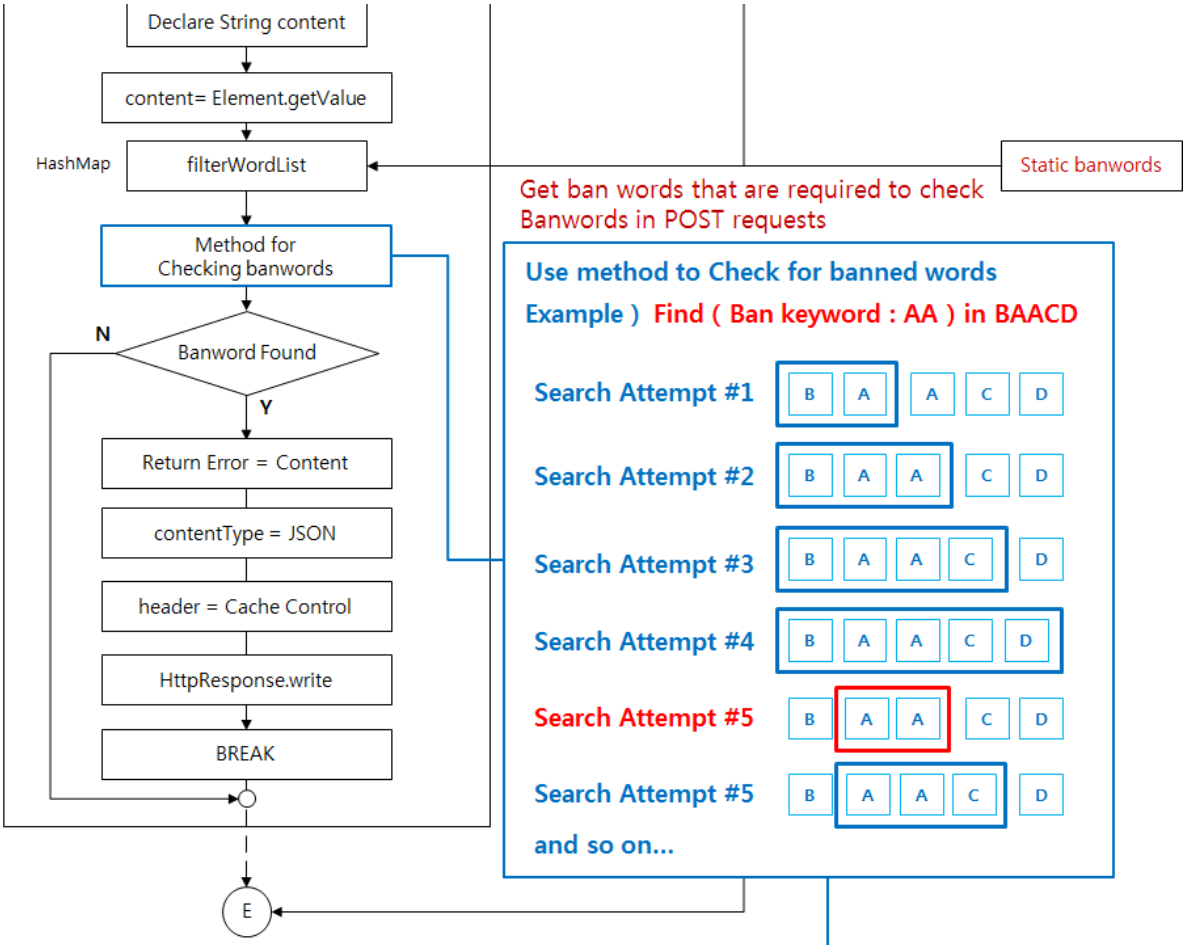
01. Filter for banwords

① Check the POST request JSON data for banwords

② Retrieve the banwords from the static data which are saved in a String

③ Find banned words mixed in words and sentences.



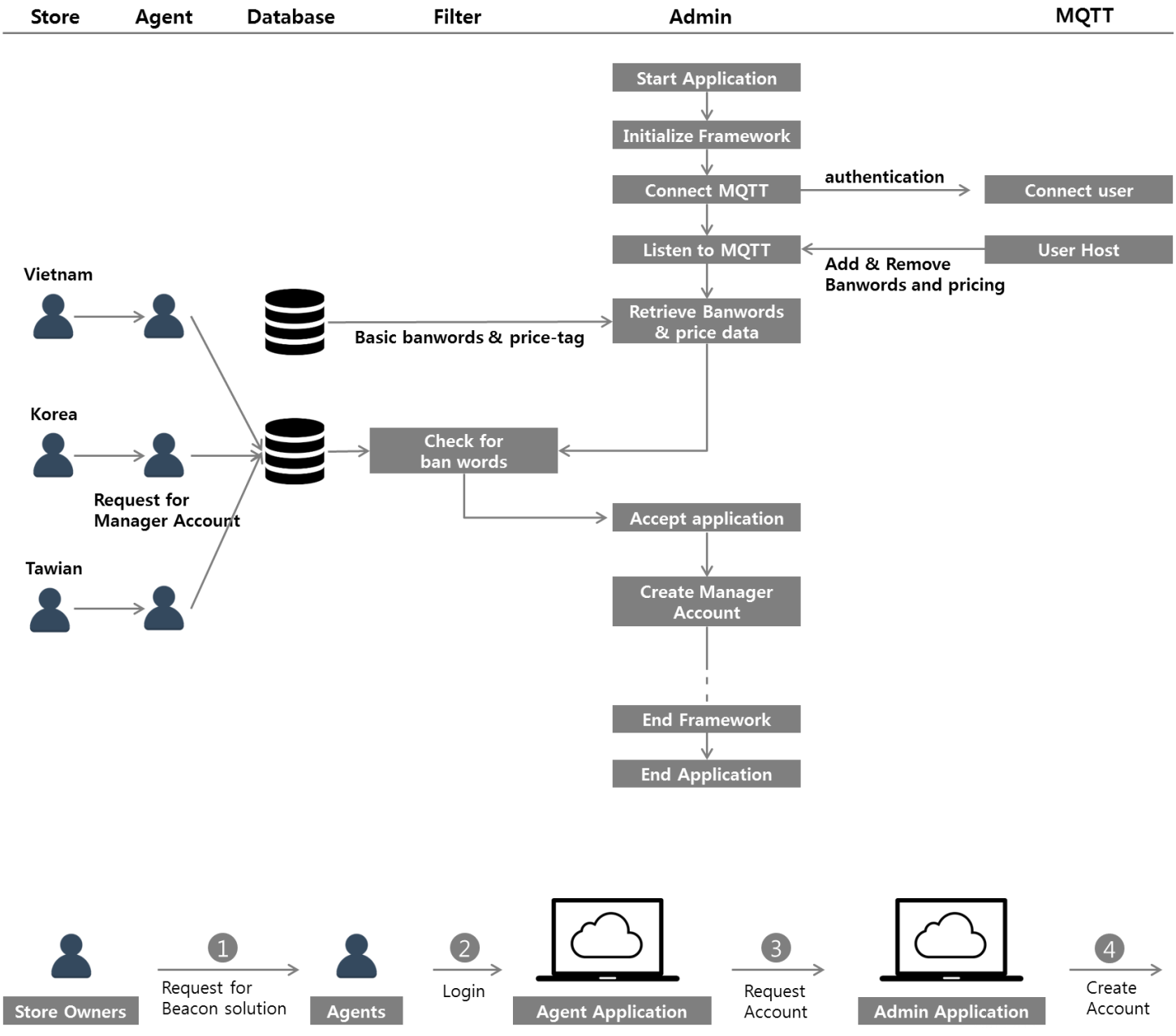



```java
public static String checkFilterWord4HashMap(HashMap _filterWordList, String _contents) {

    String content = null;
    HashMap<String, String> hash = new HashMap<String, String>();

    if (_contents == null)
        return null;
    content = _contents.toLowerCase();

    StringTokenizer st = new StringTokenizer(content, DELIMITERS, false);


    while (st.hasMoreElements()) {
        String str = st.nextToken();
        int length = str.length();
        String substr;

        String tmpstr = hash.get(str);
        if (tmpstr != null)
            continue;

        if (Common.getInstance().getServletProp("banWord.filter.containsFlag").toLowerCase().equals("true")) {
            for (int k = 0; k < length - FILTER_WORD_MIN_LENGTH + 1; k++) {
                for (int i = FILTER_WORD_MIN_LENGTH + k; i <= length && i < FILTER_WORD_MAX_LENGTH; i++) {
                    substr = str.substring(k, i);
                    if (_filterWordList.get(substr.toLowerCase()) != null)
                        return substr;
                }
            }
        } else {
            if (_filterWordList.get(str.toLowerCase()) != null)
                return str;
        }

        hash.put(str, str);
    }

    return null;
}
```

Admin Application

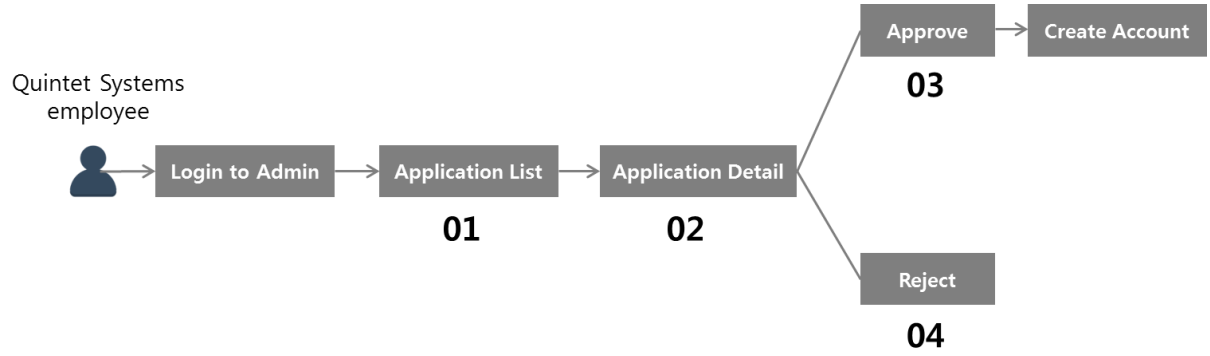
The Lifecycle

The admin is a web-based application which is used by the employees of Quintet Systems. When the agents send their application forms to the Quintet Systems Database, the employers will either admit or decline the application form. During the process the admin application will consistently filter out the banwords because some countries like Vietnam ban words that might harm the reputation of the communist regime. Newly added banwords & price-tags will be recieved by the MQTT server, and will be added to the list of static components of the framework.



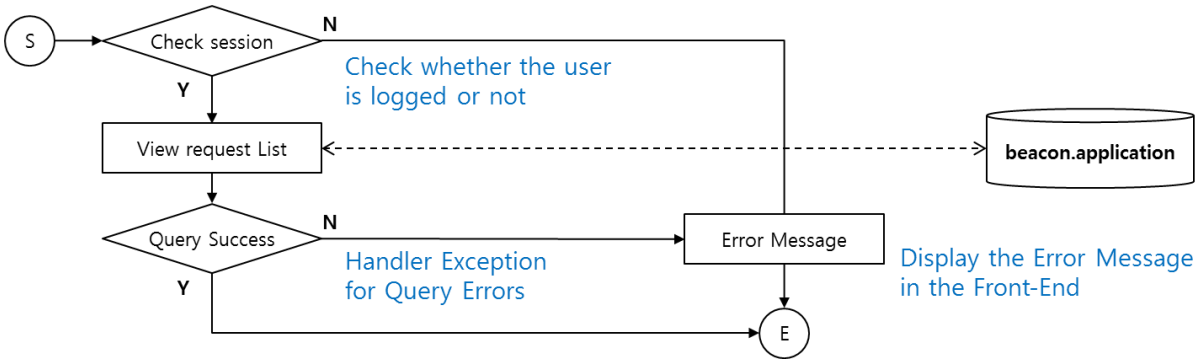
The Scenario

The main user of the Admin application is Quintet Systems employees who are approving the application made from the agents. The employees may deny or accept the application and the agents may check their status through their agent application. Since the application request is connected to future purchase of services in the Manager application, the Admin application should check for proper encrypted codes in order to confirm the valid purchase-related request from the client. (encrypted codes are created from basic Mysql encryption)



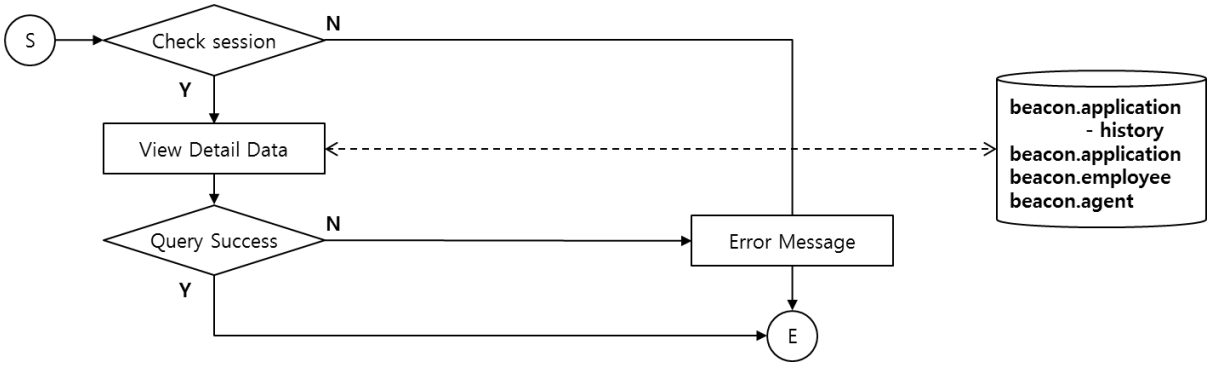
01. Application List

① Applied the basic principles of MVC pattern for the development



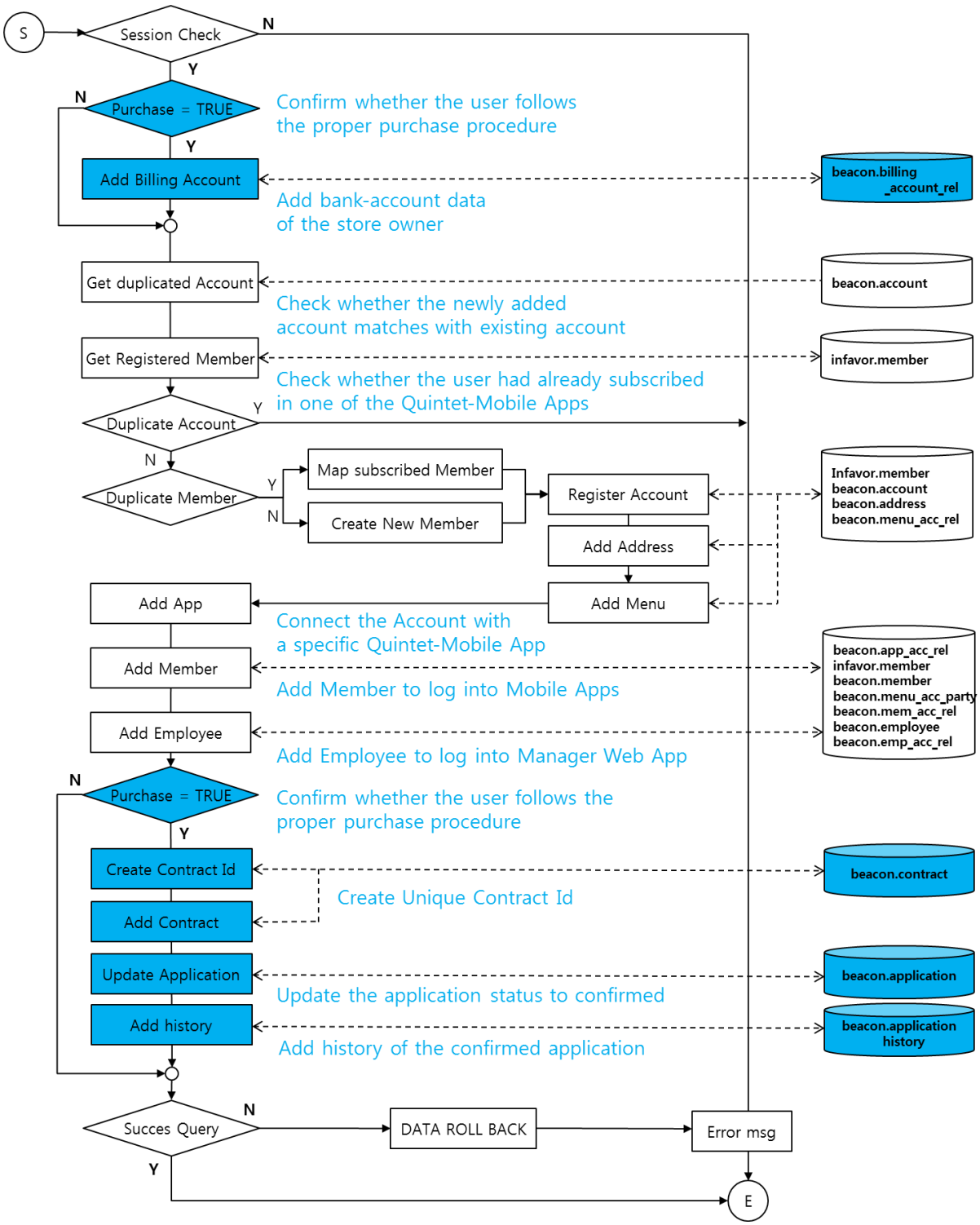
02. Application Detail

① Applied the basic principles of MVC pattern with Join Query



03. Approve Application

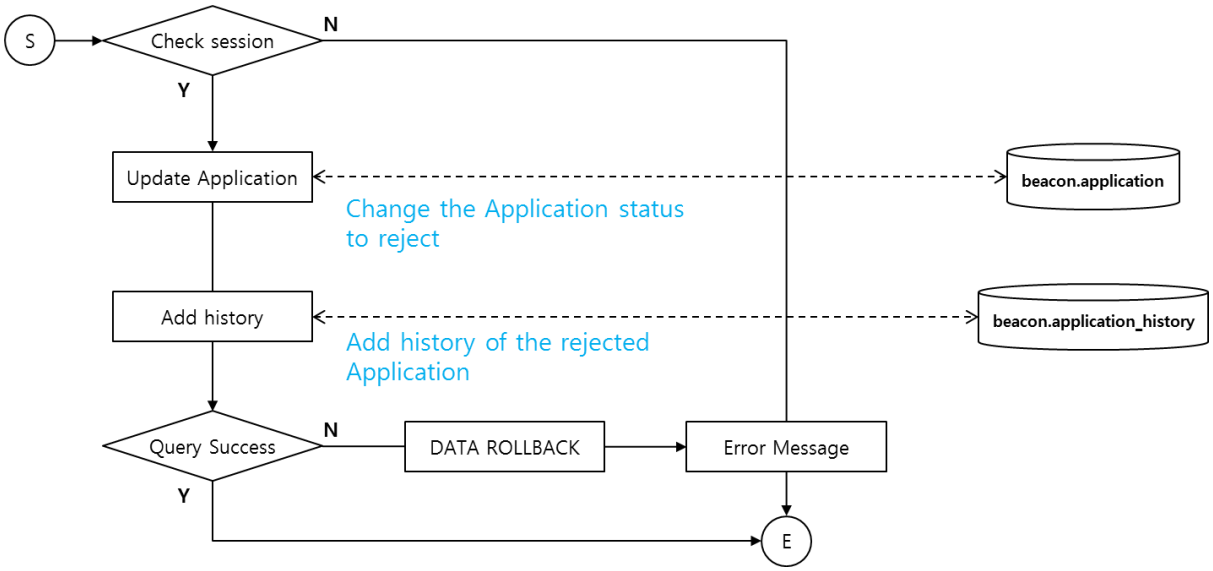
① Check the encrypted data from the client to confirm the valid purchase-related request
② Commit the data only when all the insert and update queries are successful
③ Use Select-Insert in Queries in order to avoid using data from client
④ Add Contract data with the account in order to track history of future purchases
⑤ Avoid using independent select queries during the middle of start-commit transaction to prevent deadlocks caused by delayed select queries



04. Reject Application

① Use Join-Update in Queries in order to avoid using data from client

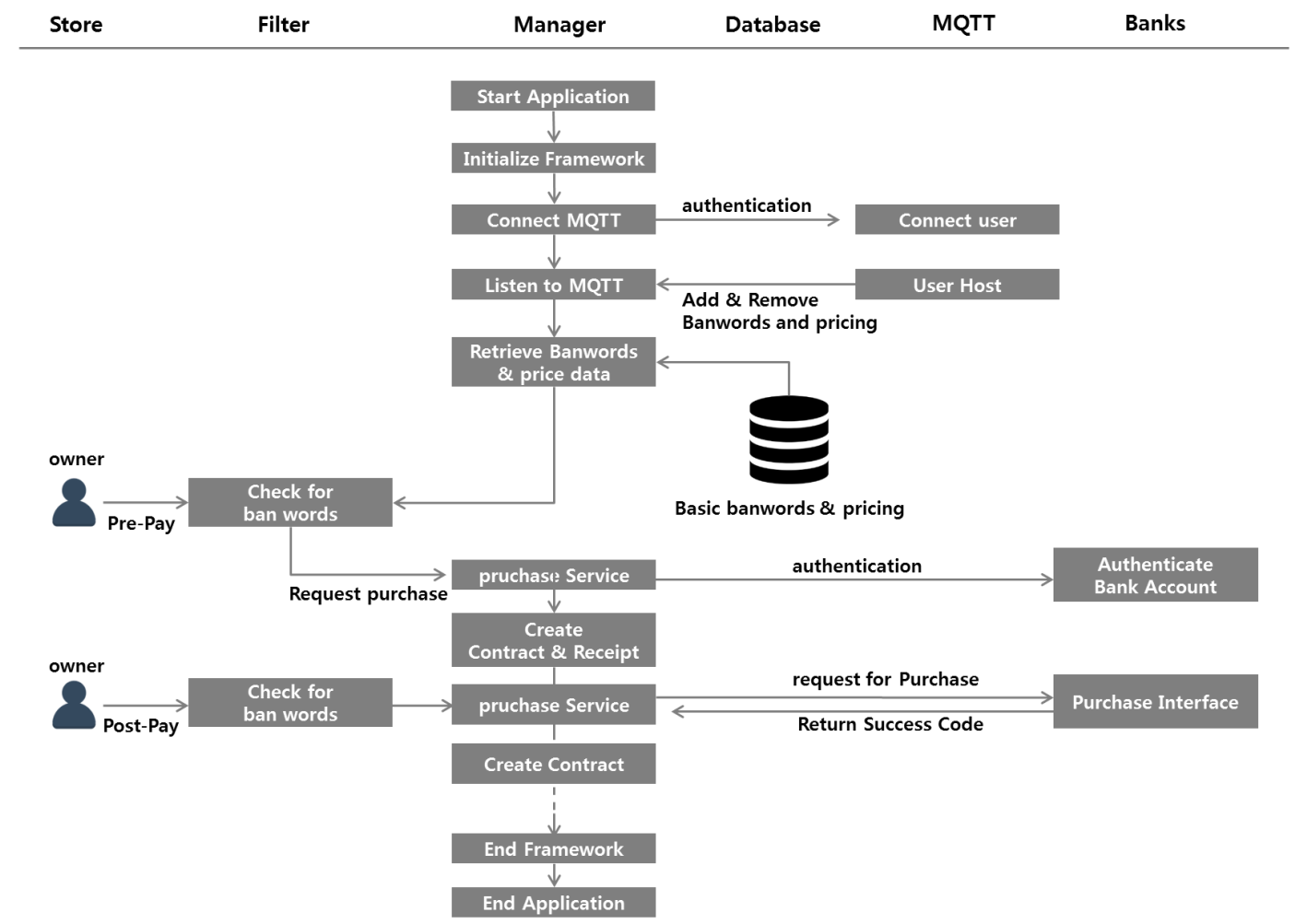
② Notify the rejection to agents and store owners by updating the application history



Manager Application

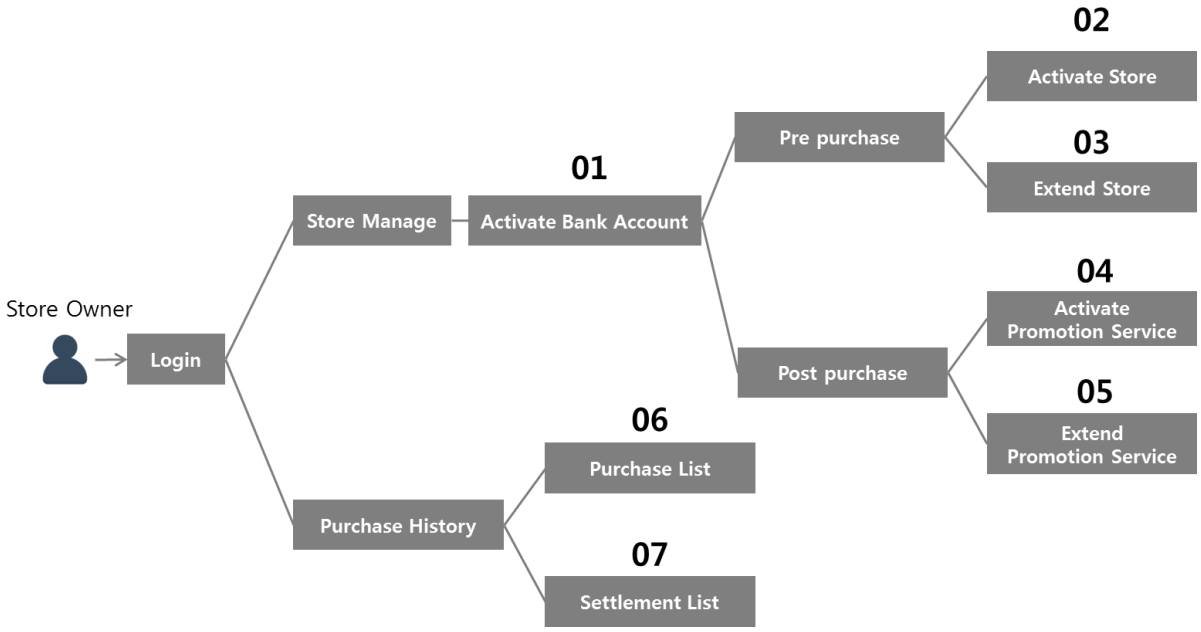
The Lifecycle

The manager is a web-based application which is used by the store owners who have been approved by the employees of Quintet Systems. The store owners may create stores to advertise their stores with the owner's customized-App or Quintet-System's App. The store owners may also expand their promotions by using IOT integrated coupons and stamps. In order to use the features of the manager the users should first pre-purchase a store and post-purchase additional services.



The Manager Application

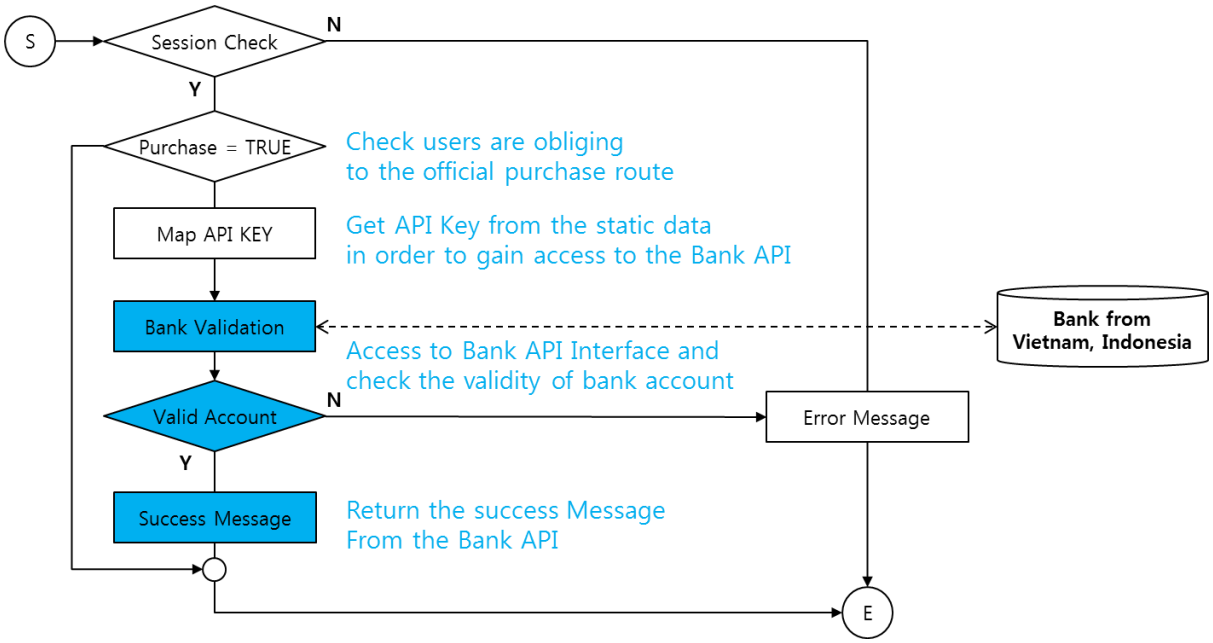
The main user of the Manager application is the store owners who have been approved by Quintet Systems employees through the Admin application. The store owners may pre-pay (create one store which is displayed in the mobile-app) and post-pay for the services (coupons or stamps which can be used as a promotion tool when the users interacts with IOT devices). The Manager application should check for proper encrypted codes in order to confirm the valid purchase-related request from the client. (encrypted codes are created from basic Mysql encryption)



01. Activate Bank Account

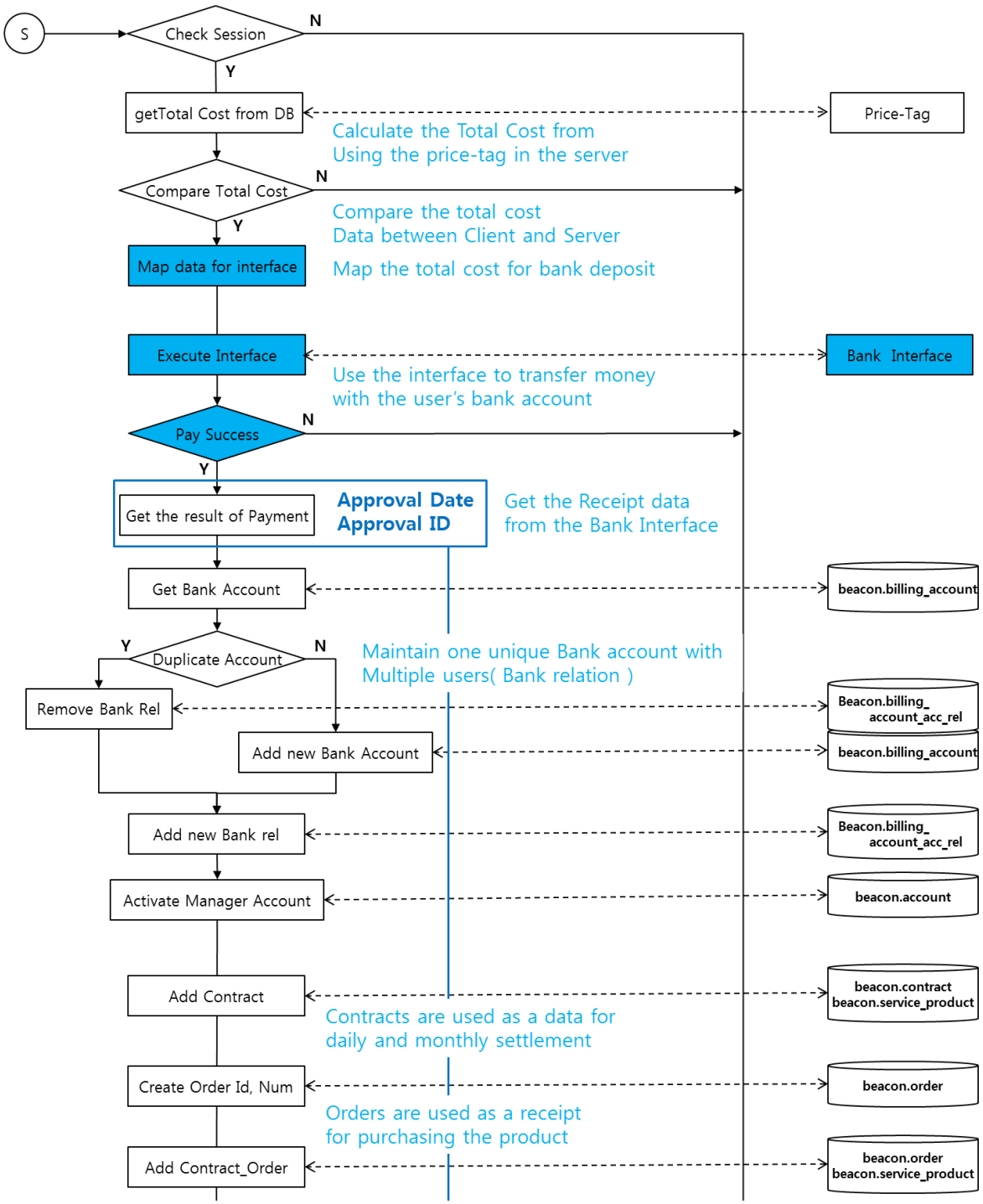
① Validate the bank account by using the bank interface

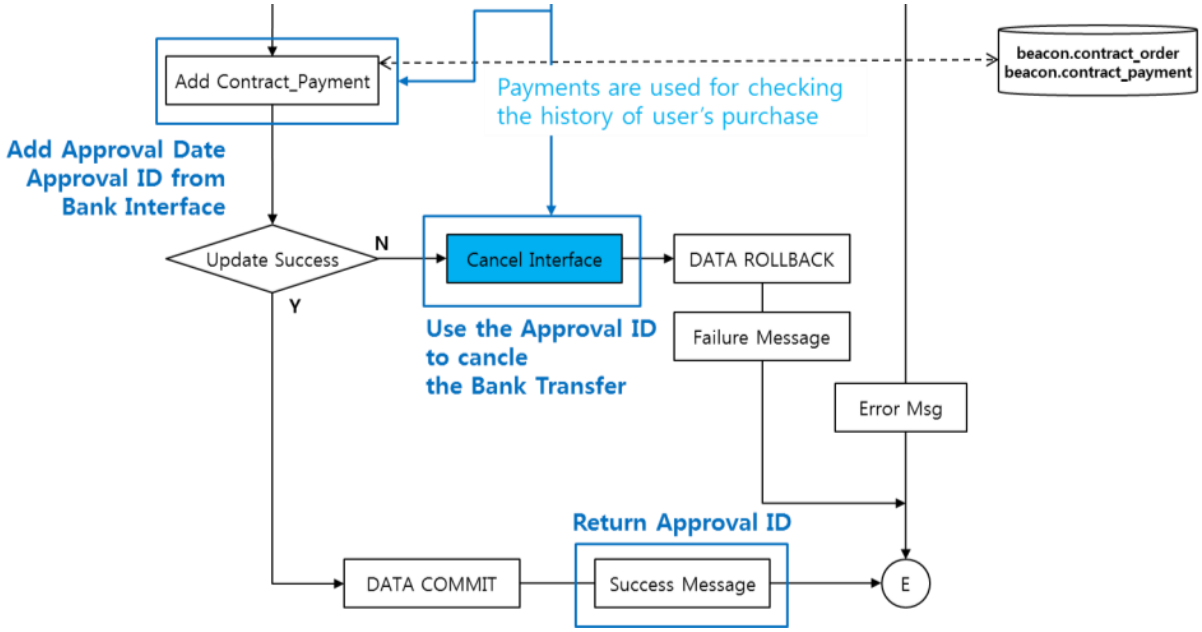
② Use the API key in the static components in order to gain access to Bank Interfaces



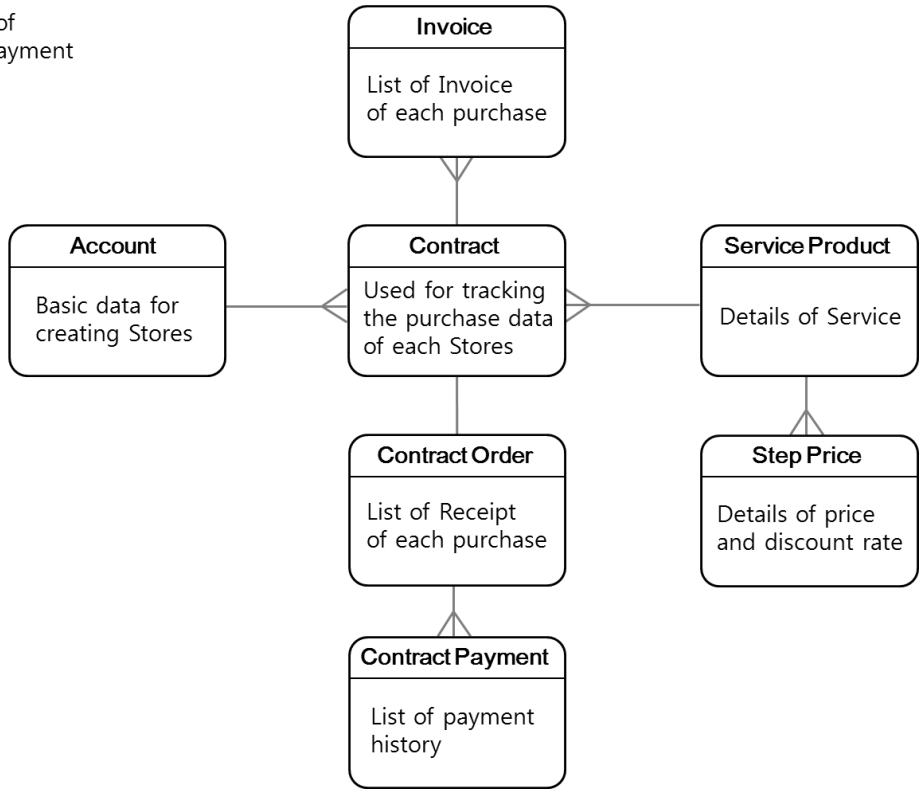
02. Pre Purchase- Create Store

① Check the validity of client's bank-account data
② Add the pre-purchase as the main-contract
③ Store the approval code used for canceling the bank deposit, if errors occur during the purchase procedure
④ Save the user's bank account in the database for future purchases
⑤ Avoid using independent select-queries during the middle of start-commit transaction to prevent deadlocks caused by delayed select queries





Database Relationship of
Contracts, Order and Payment



02. Pre Purchase- Request Extension

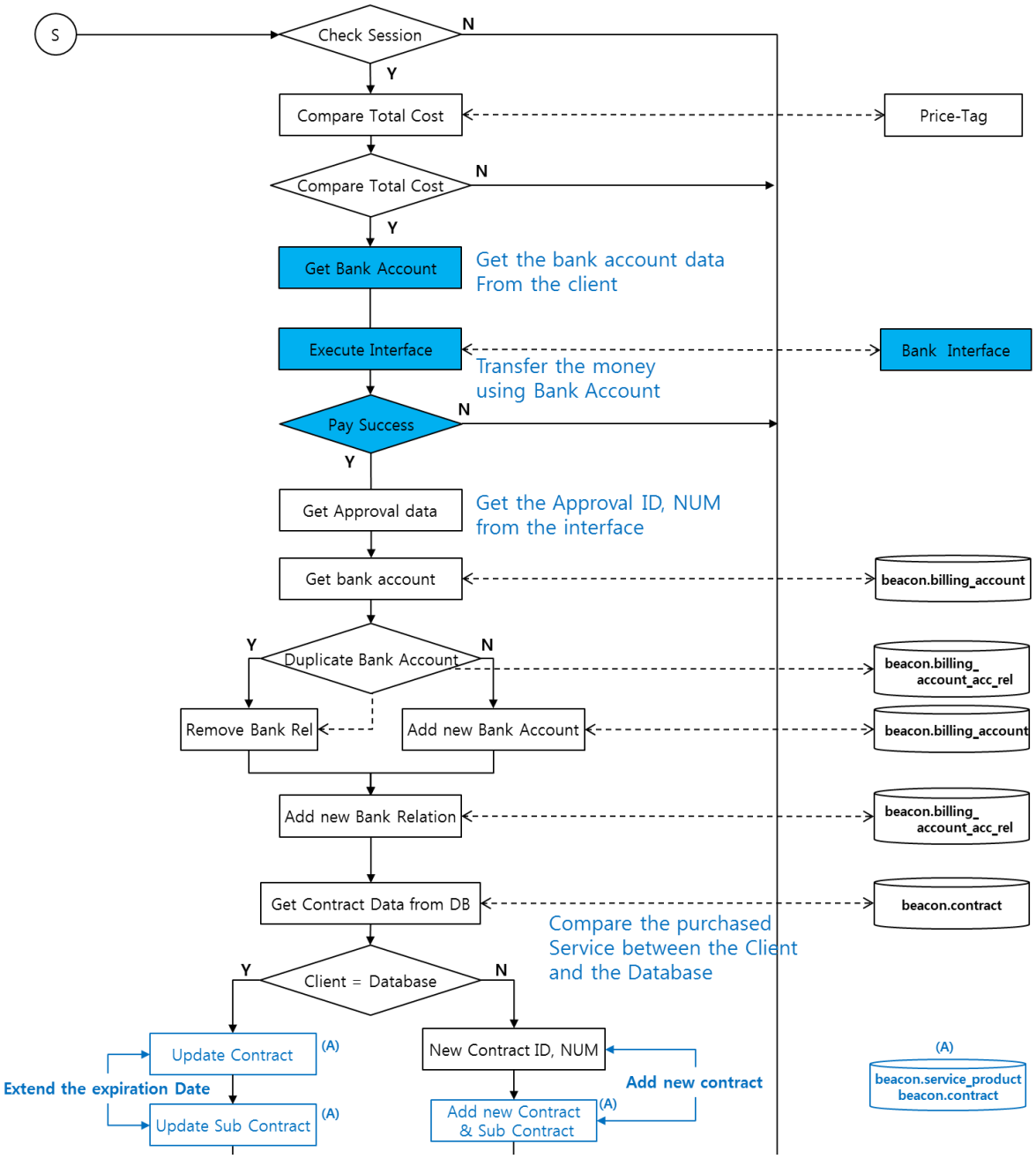
① Compare the total cost between the client and the server for possible errors

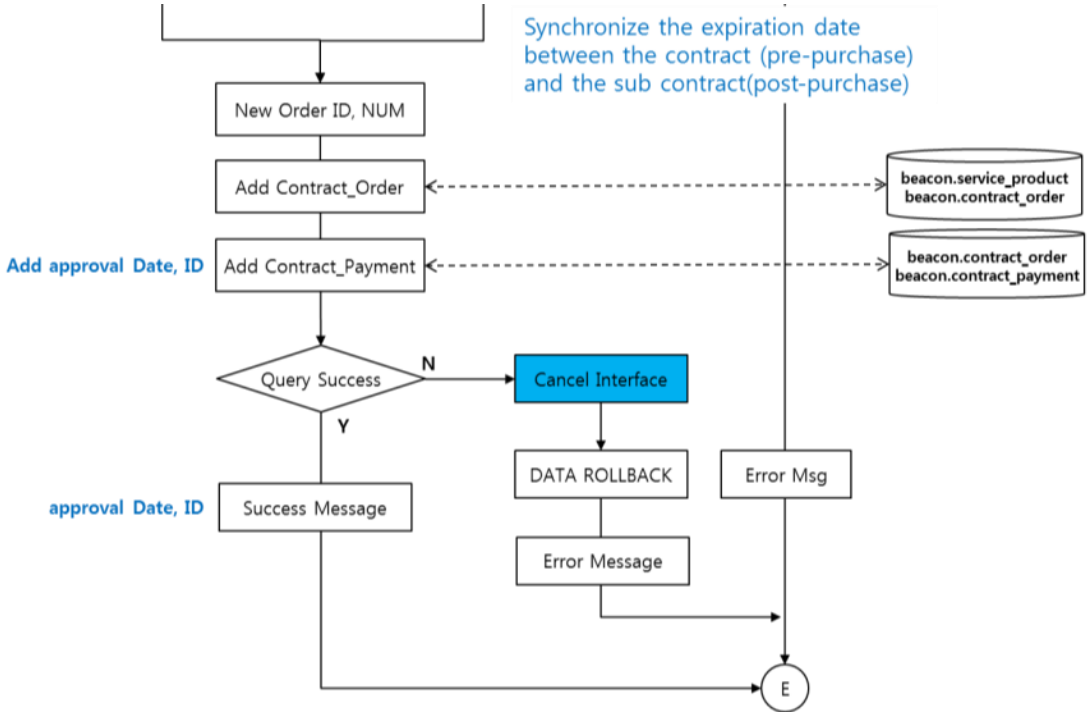
② Add the pre-purchase as the main-contract and post-purchase as the sub-contract

③ Store the approval code to cancel the bank deposit in case there are errors found during the purchase procedure

④ Check for changes in the types of purchase products in order to decide whether to issue a new contract or extend the contract

⑤ Synchronize the expiration date of the Contract (pre-purchase) and the Sub-Contract (post-purchase).




Synchronize the expiration date
between the contract (pre-purchase)
and the sub contract(post-purchase)
New Order ID, NUM
Add Contract_Order
beacon.service_product
beacon.contract_order
Add approval Date, ID
Add Contract_Payment
beacon.contract_order
beacon.contract_payment
Query Success
N
Cancel Interface
Y
DATA ROLLBACK
Error Msg
approval Date, ID
Success Message
Error Message
E

03. Post Purchase- Add Promotion Services to the Store

① Initiate Start-Commit transaction of Insert-queries after the images are successfully uploaded

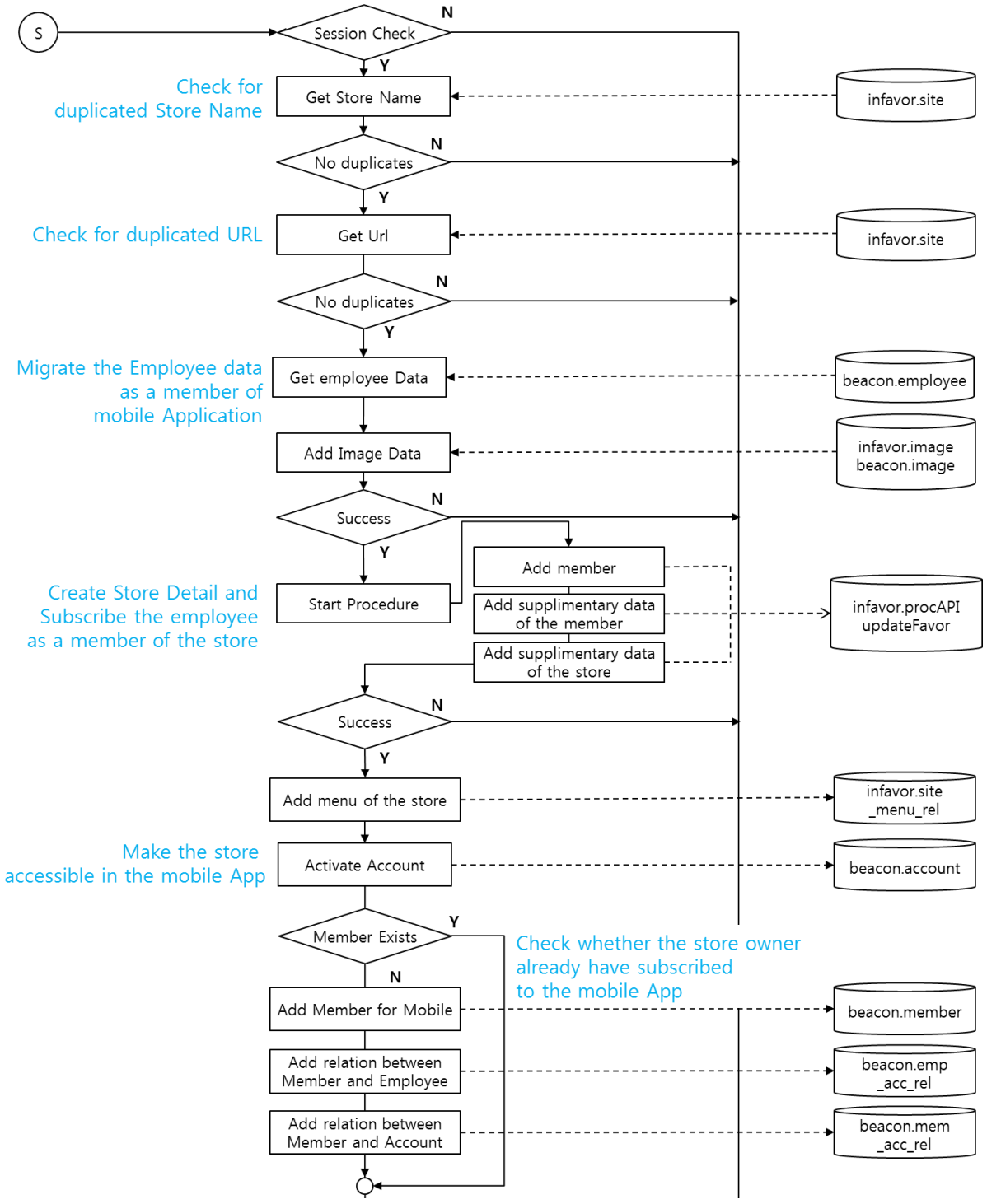
② Group all insert queries that are related to the mobile app with the procedure

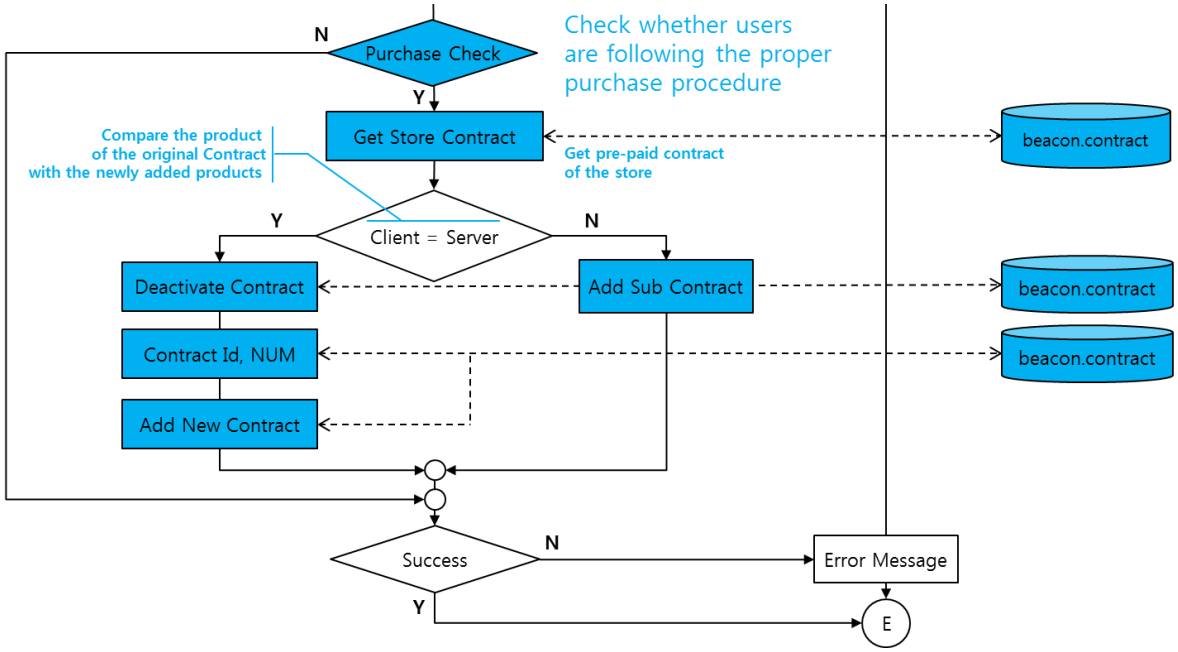
③ Add the post purchase as the sub contract

④ Check for differences between the original service and the newly added services of the main contract

If different > deactivate the original contract (main-contract) and add a new contract

If same > add sub contracts (sub-contract) in the original contract




Purchase Check
N
Y
Check whether users are following the proper purchase procedure
Get Store Contract
Get pre-paid contract of the store
beacon.contract
Compare the product of the original Contract with the newly added products
Client = Server
Y
N
Deactivate Contract
Add Sub Contract
beacon.contract
Contract Id, NUM
beacon.contract
Add New Contract
Success
N
Error Message
Y
E

04. Post Purchase- Add or Quit Additional Services

① Check compromised contract data about client

② Check for the updates of the store from other logged users in order to avoid confusion

③ Check for the changes in all the contract

If changes exist> procede with the update

If no changes > skip the update

⑤ Check for differences between the original services and the newly added services in the main contract

If different > add new main & sub contract

If not different > update main & sub contract

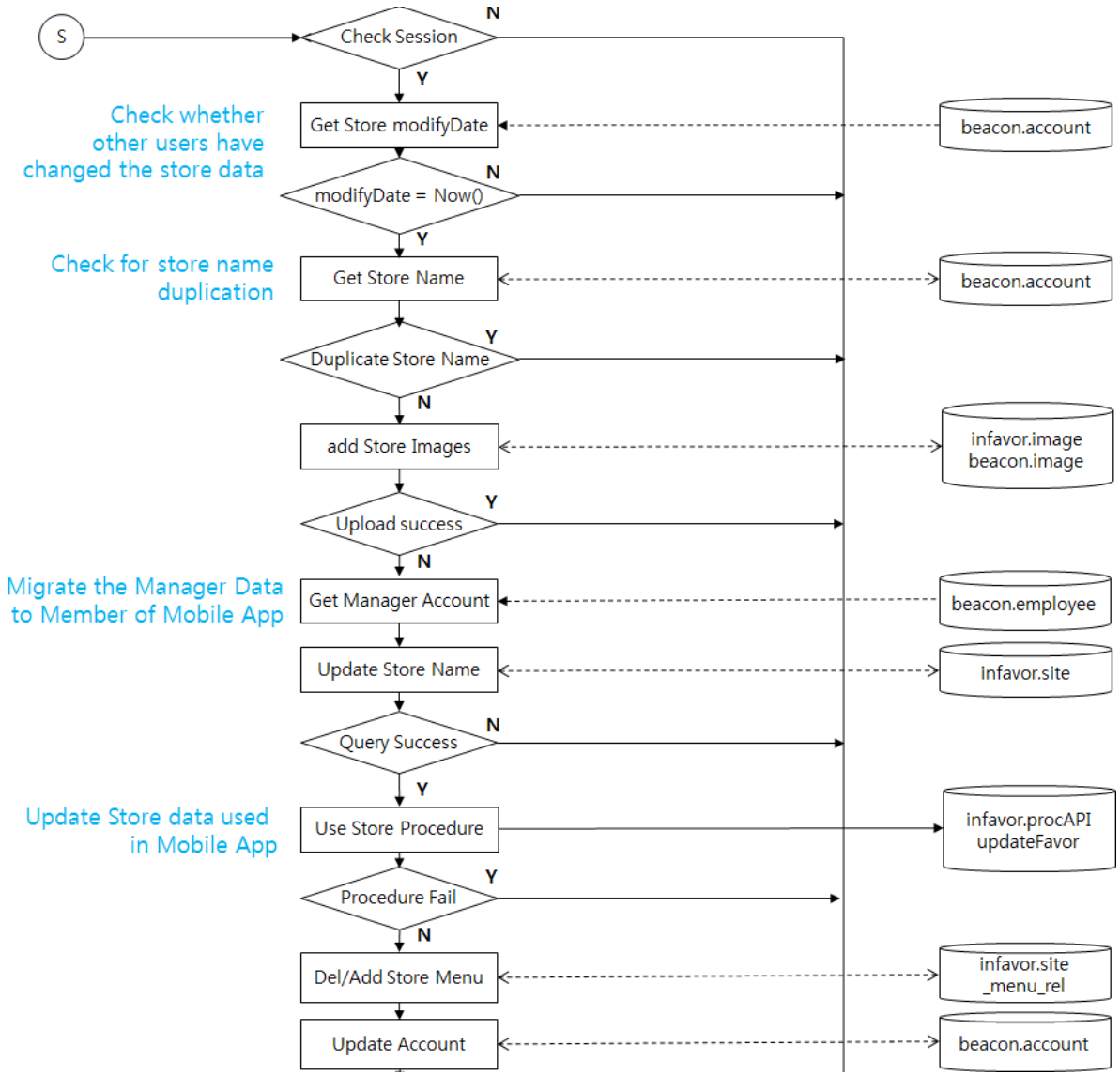
④ Check for differences between the original services and the newly added services in the sub contract

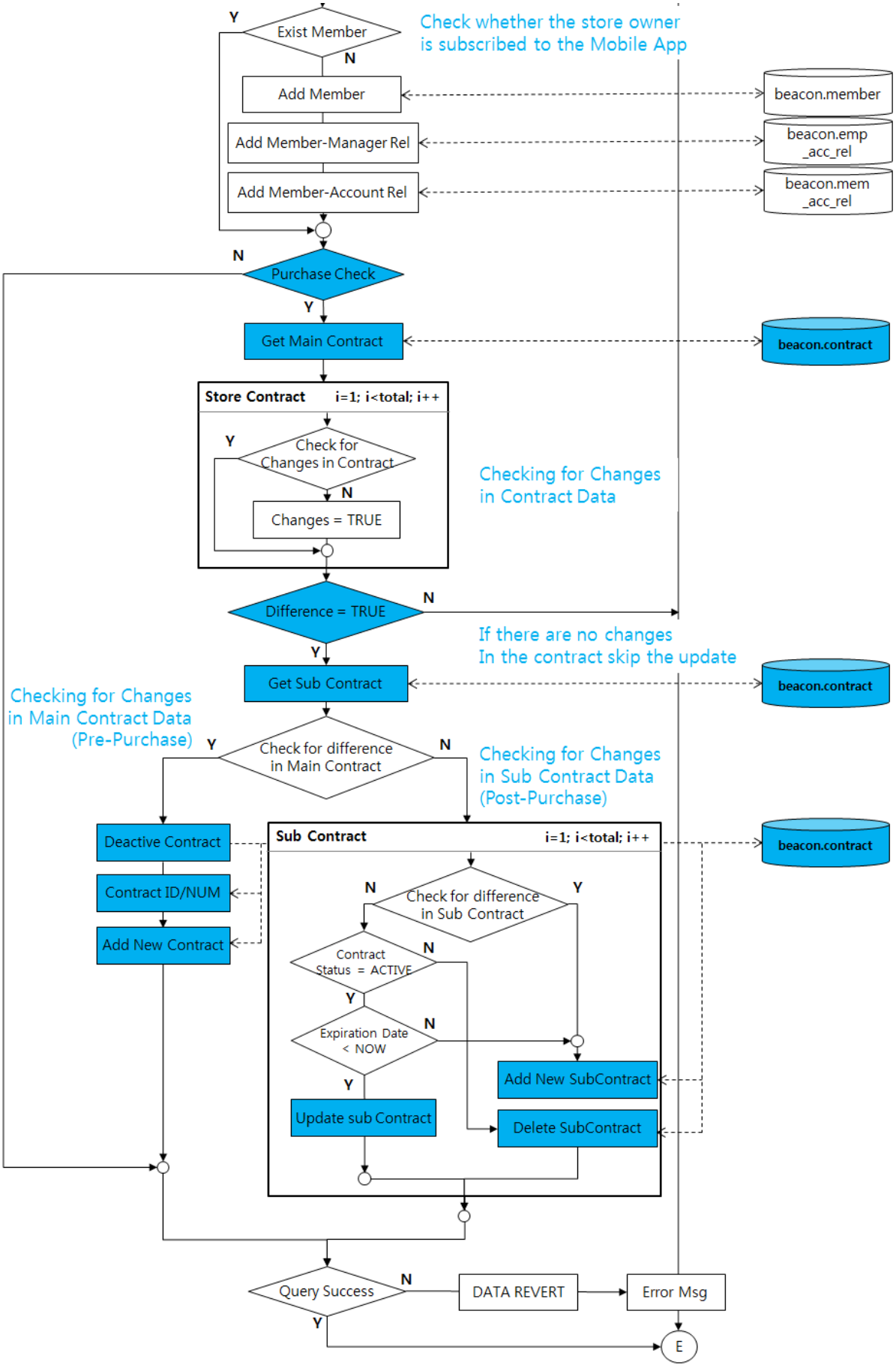
If different > add new sub contract

⑤ Check for status and expiration date of sub contract

If status or expiration date are expired > add new sub contract

If status or expiration date are valid > update sub contract

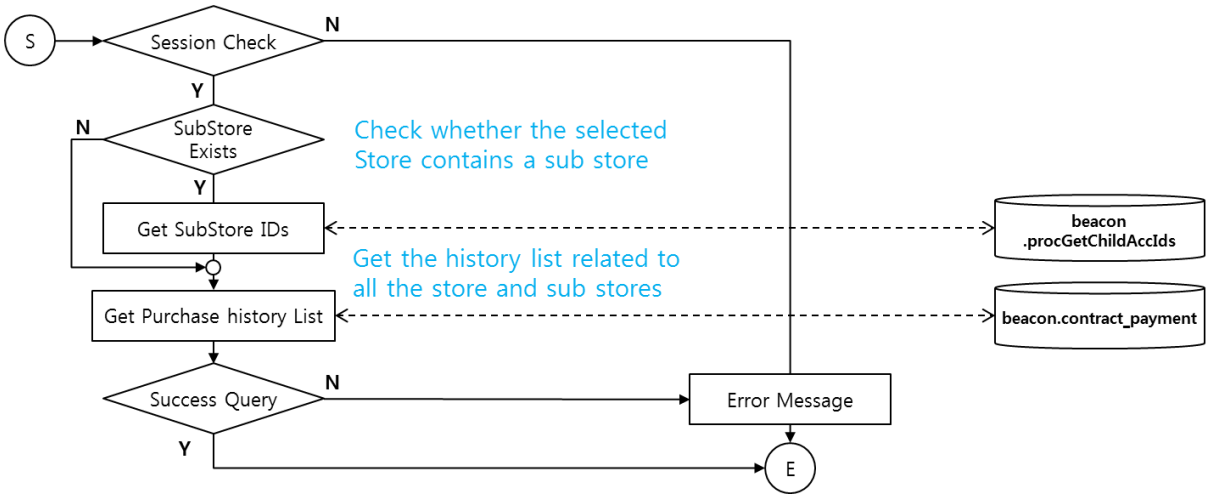



Exist Member
Y
N
Check whether the store owner is subscribed to the Mobile App
Add Member
Add Member-Manager Rel
Add Member-Account Rel
beacon.member
beacon.emp_acc_rel
beacon.mem_acc_rel
Purchase Check
N
Y
Get Main Contract
beacon.contract
Store Contract i=1; i<total; i++
Check for Changes in Contract
Changes = TRUE
Checking for Changes in Contract Data
Difference = TRUE
If there are no changes In the contract skip the update
Get Sub Contract
Checking for Changes in Main Contract Data (Pre-Purchase)
Check for difference in Main Contract
Checking for Changes in Sub Contract Data (Post-Purchase)
Deactive Contract
Contract ID/NUM
Add New Contract
Sub Contract i=1; i<total; i++
Check for difference in Sub Contract
Contract Status = ACTIVE
Expiration Date < NOW
Add New SubContract
Update sub Contract
Delete SubContract
Query Success
DATA REVERT
Error Msg
E

05. Purchase List

① Applied the basic principles of MVC pattern for the development

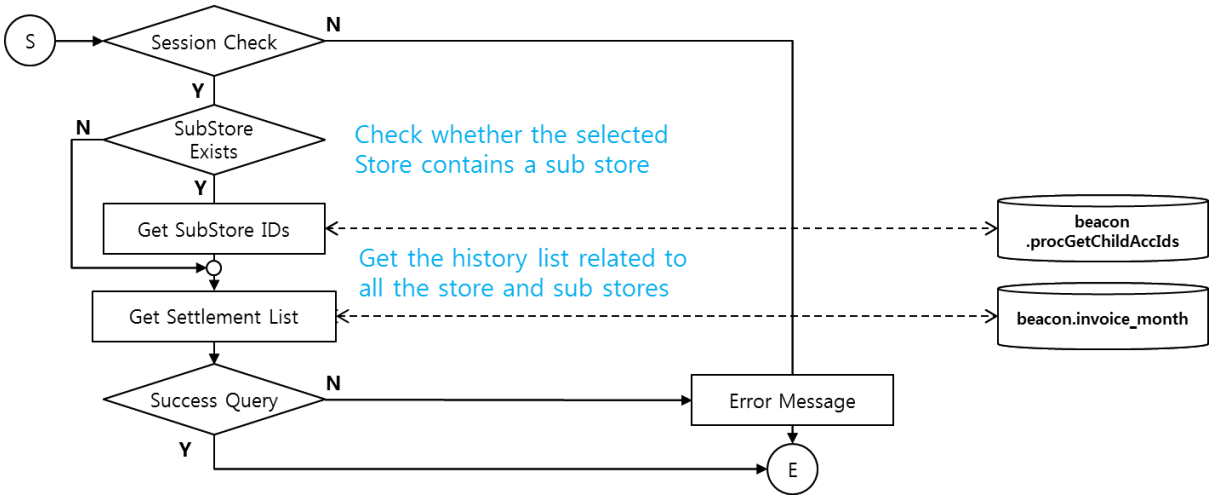
② The selected Store should get all the purchase history of it's sub-stores



06. Settlement List

① Applied the basic principles of MVC pattern for the development

② The selected Store should get all the settlement history of it's sub-stores



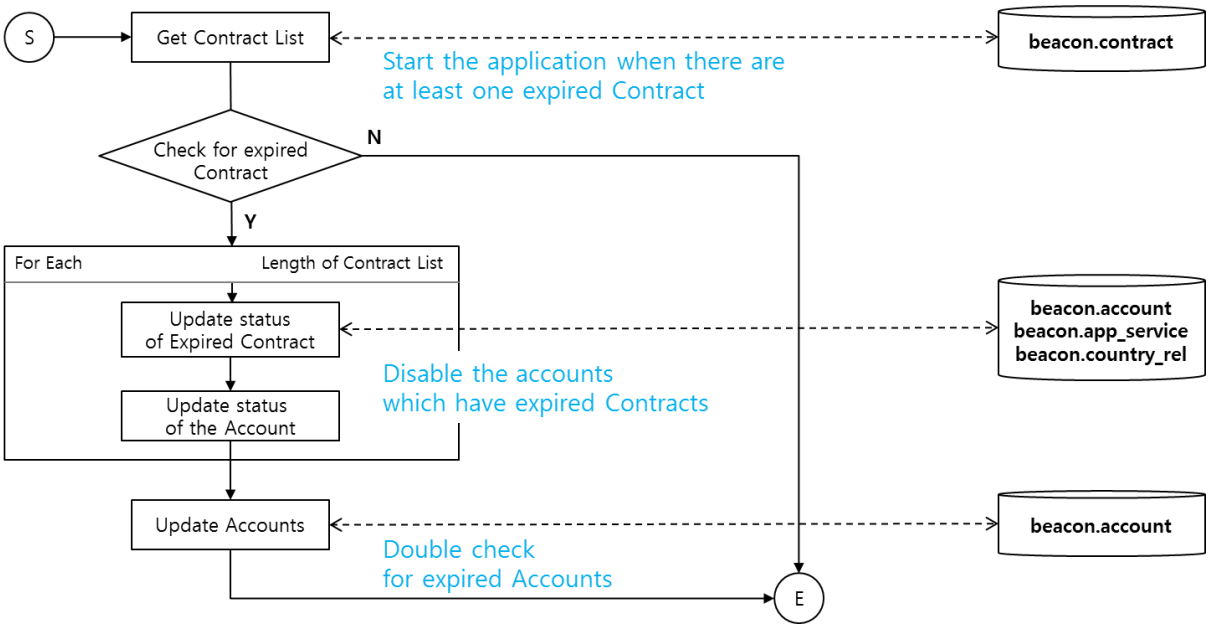
The Batch Application

The main purpose of the batch is to add up the post-paid purchase (Sub-Contract) and transfer the total cost into settlement data, and also disable the Stores (accounts) with expired contracts. The purchase-related objects in the batch application are scheduled to run daily.

01. Contract Expiration Check

① Sort out the expired contract data and disable the account(Store)

② Activate the application daily

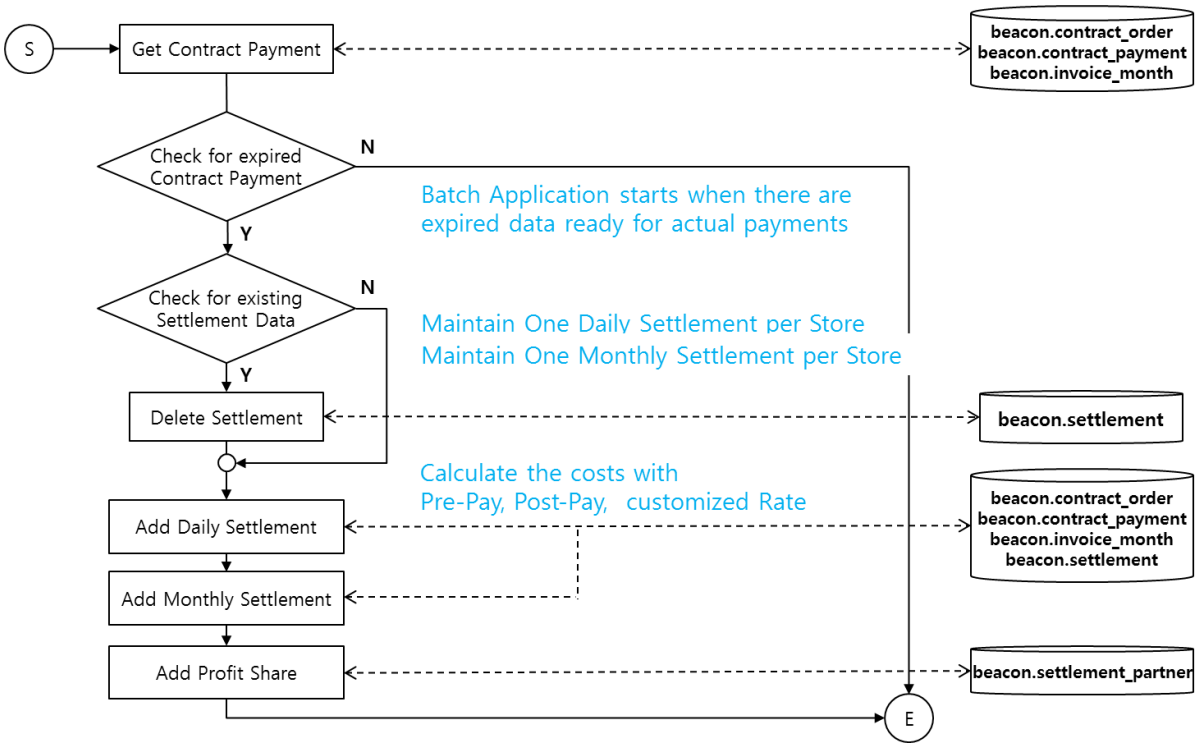


I. Sample Query for disabling Stores (accounts)

```
update beacon.account a
join beacon.app_service_country_rel ascr
on a.APP_SERVICE_TYPE = ascr.ID
join beacon.contract c on a.ID = c.ACCOUNT_ID    each Contracts of the Stores
set
    a.MODIFY_BY = beacon.getSuperUserId()
  , a.MODIFY_DATE = now()
  , a.FLAG = 0    disabling the Store
where a.FLAG = 1
and a.STATUS = beacon.groupCodeCodeName('PARTY_STATUS', 'HALT', pLang)
and adddate(coalesce(c.END_DATE, cast(now() as date)),
coalesce(ascr.PROBATION_DAY, 0)) <= now()    check for expiration date
```

02. Settlement Check

① Sort out the contract payment data and calculate them into daily settlements
② Sort out the daily settlement data and calculate them into monthly settlements
③ Check the invoice status to sort out the un-paid payment of stores
④ Calculate the final profits by referring to the customized discount-rates



I. Sample Query for Daily Store Settlement

① POSTPAID_AMT: price data collected from the invoice table
② PREPAID_AMT: price data collected from the payment_history table
③ SETTLEMENT AMT: (POSTPAID + PREPAID) * AGENT_SHARE
④ Agent share are customized for each country's policy

```
INSERT INTO `beacon`.`settlement_store_day`
(`ID`,
`CREATE_BY`,
`MODIFY_BY`,
`CREATE_DATE`,
`MODIFY_DATE`,
`TYPE`,
`STATUS`,
`FLAG`,
`SETTLEMENT_DATE`,
`POSTPAID_AMT`,      daily post-payment collected from the invoice
`PREPAID_AMT`,       daily pre-payment collected from the payment history
```

```
`TOTAL_AMT`,
`SETTLEMENT_AMT`, daily settlement calculated with the customized discount rate
`REVENUE_SHARE_RATE`,
`ACCOUNT_ID`,
`AGENT_ID`)
SELECT beacon.getNewID() AS ID
            , beacon.getSuperUserId() AS CREATE_BY
            , beacon.getSuperUserId() AS MODIFY_BY
            , now() AS CREATE_DATE
            , now() AS MODIFY_DATE
            , beacon.codeName('SETTLEMENT_DAY_TYPE', 'BATCH') AS TYPE
            , beacon.codeName('SETTLEMENT_DAY_STATUS', 'FINISHED') AS STATUS
            , 1 as FLAG
            , cast(pSettlementDate as date) as 'SETTLEMENT_DATE'
            , sum(COALESCE(im.INVOICE_AMT, 0)) as 'POSTPAID_AMT'
            , sum(COALESCE(cp.PAYMENT_AMT, 0)) as 'PREPAID_AMT'
            , sum(COALESCE(im.INVOICE_AMT,0))
             +sum(COALESCE(cp.PAYMENT_AMT, 0)) as 'TOTAL_AMT'
            , (sum(COALESCE(im.INVOICE_AMT,0))
             +sum(COALESCE(cp.PAYMENT_AMT, 0)))
             *ag.SHARE_RATE as 'SETTLEMENT_AMT'
            , ag.SHARE_RATE as 'REVENUE_SHARE_RATE'
            , ac.ID as 'ACCOUNT_ID'
            , ag.ID as 'AGENT_ID'
from beacon.account ac
join beacon.agent ag on ag.ID = ac.AGENT_ID
join beacon.contract c on ac.ID = c.ACCOUNT_ID
left outer join beacon.contract_order co on c.ID = co.CONTRACT_ID
left outer join beacon.contract_payment cp on cp.CONTRACT_ORDER_ID = co.ID
and cast(cp.APPROVAL_DATE as date) = cast(pSettlementDate as date)
and cp.STATUS = beacon.codeCodeName('PAYMENT_STATUS', 'PAID')
left outer join beacon.invoice_month im on ac.ID = im.ACCOUNT_ID
and im.STATUS = beacon.codeCodeName('INVOICE_STATUS', 'RECEIPT')
and cast(im.RECEIPT_DATE as date) = cast(pSettlementDate as date)
group by ac.ID
```

II. Sample Query for Monthly Store Settlement

① POSTPAID_AMT: post-paid data collected from the daily_settlement table
② PREPAID_AMT: pre-paid data collected from the daily_settlement table
③ SETTLEMENT AMT: settlement data from the daily_settlement table
④ NONPAYMENT_AMT: collected data which is not paid from the invoice table

```
INSERT INTO `beacon`.`settlement_store_month`
(`ID`,
`CREATE_BY`,
`MODIFY_BY`,
`CREATE_DATE`,
`MODIFY_DATE`,
`TYPE`,
`STATUS`,
`FLAG`,
`SETTLEMENT_MONTH`,
`POSTPAID_AMT`,          monthly cost collected from post-payment daily settlement
`PREPAID_AMT`,           monthly cost collected from pre-payment of daily settlement
`NONPAYMEMT_AMT`,        monthly cost of unpaid payment from invoice
`TOTAL_AMT`,
`SETTLEMENT_AMT`,        monthly cost with customized discount rate from daily settlement
`REVENUE_SHARE_RATE`,
`ACCOUNT_ID`,
`AGENT_ID`)
SELECT beacon.getNewID() AS ID
        , beacon.getSuperUserId() AS CREATE_BY
        , beacon.getSuperUserId() AS MODIFY_BY
        , now() AS CREATE_DATE
        , now() AS MODIFY_DATE
        , beacon.codeName('SETTLEMENT_TYPE', 'BATCH') AS TYPE
        , beacon.codeName('SETTLEMENT_STATUS', 'BATCH_FINISHED') AS STATUS
        , 1 as FLAG
        , pSettlementMonth
        , sum(ssd.POSTPAID_AMT) as 'POSTPAID_AMT'
        , sum(ssd.PREPAID_AMT) as 'PREPAID_AMT'
        , sum(im.INVOICE_AMT) as 'NONPAYMEMT_AMT'
        , sum(ssd.TOTAL_AMT) as 'TOTAL_AMT'
        , sum(ssd.SETTLEMENT_AMT) as 'SETTLEMENT_AMT'
        , ssd.REVENUE_SHARE_RATE as 'REVENUE_SHARE_RATE'
        , ssd.ACCOUNT_ID as 'ACCOUNT_ID'
```

```
            , ssd.AGENT_ID as 'AGENT_ID'
from beacon.settlement_store_day ssd
left outer join beacon.invoice_month im on ssd.account_id = im.account_id
and im.STATUS in ( beacon.codeName('INVOICE_STATUS', 'NONPAYMENT')
                        , beacon.c odeName('INVOICE_STATUS', 'LONGNONPAYMENT')
where cast(concat(pSettlementMonth,'01') as date) <= ssd.SETTLEMENT_DATE
and ssd.SETTLEMENT_DATE > DATE_ADD(cast(concat(pSettlementMonth,'01') as date)
    , interval -1 MONTH )
group by ssd.ACCOUNT_ID
```