Two models of data exchange.

* For batch exchange
* For instantaneous communication

Batch communication : This is used for Inward order receiving and day end report Outward submissions

Day Start/ End.:

* This activity may occur at start of day {prior to 9:00 AM or 9:00 PM} so our batch should be tuned to process this during this time
* Alternatively there could be data push every hour if there is a need for replenishments, in which case there has to be a polling mechanism or trigger-based mechanism to initiate the parsing of the file as and when it is received
* If there are no files pushed till 9:30 AM for that day, there should be an alert to system.
* There should also be an acknowledgement system to confirm that the file was received and data parsed successfully.

Tasks

* **Day Start**

1. Create a drop zone/folder where a client can publish / push a file of type {csv/Json} {This is one for each client/aggregator}
2. Data structure needs to be flexible to parse any number of fields
3. Parse it and load DB {in most likelihood we would share the structure to aggregator but let’s keep it expandable}
4. The data would contain all details as mentioned in the SKU structure.
5. Save the date & time for inward order that was received.
6. Parse the same and store in our master DB
7. Sort the order by location and push to remote location DB, both at server and remote DB levels
8. Update and test if its reflecting across all DB’s

**Replenish orders** : Aggregators might send inward order details to our system in the event of menu item getting sold earlier than expected or for dinner menu’s during the course of the day.

In such cases the system should keep polling for additional files/data in the drop zone folder for data as mentioned in the Activity section (above) and repeat the tasks as defined in Day start, however this should not impact existing data and should always append the data with the current date inward order.

* Day End
* Generate the reports for the following, sorted by client/aggregator.
  1. Daily Sales report.
  2. Daily Purchase or stocks received report
  3. Daily stocks in hand report
  4. Daily goods returned report by aggregator, restaurant, location, customer.
  5. Daily items returned to aggregator/restaurant report
* Publish these reports in the folder from where the client/aggregator can pull. All reports should be saved with date as suffix.
* Data / reports applicable to specific aggregator only should be posted

**Instantaneous communication.**

Create an API which will allow data to be exchanged instantaneously between client/aggregator and AGT {Arvir Global Tech}.

* **Delivery Phase**

1. Opened/Damaged packs will be marked as “Damaged”,
   1. DB should not account these for in stock for the day
   2. A separate table showing damaged should be created for the day and reflect in remote and server DB
2. Menu packs not invoiced/additional or ordered or wrongly delivered to different location will be marked as “To return”
   1. DB should not account these for in stock for the day
   2. A separate table showing damaged should be created for the day and reflect in remote and server DB
3. All “In order” menus received will be received as Inwards needs to be acknowledge as “Received”
   1. DB should **account** these for in stock for the day
   2. A separate table showing damaged should be created for the day and reflect in remote and server DB
4. Difference in Ordered/Received/ non ordered {when non ordered items are delivered} should be persisted.
5. Summary detail should be shared to client/aggregator.
   1. Not invoiced / ordered – Correct location
   2. Not invoiced / ordered – In Correct location
   3. Additional count of menu - Correct location
   4. Additional count of menu – In Correct location
6. This report should be reflected to client / Aggregator instantaneously.
   1. If Client or aggregator sends file/API update to mark the above mentioned as “To consider as Inward,
   2. Mark the Inward DB with additional status – Additional order inward , capture date and time
   3. Reflect the new inwards and update all systems
7. These updates needs to reflect in the DB which will keep it posted through the API
8. The inventory will pass into the client/aggregator DB which will then reflect in their App.
9. The details will be shared in the EOD report.
10. Remote DB and Server DB needs to update the SKU

* **User Experience : Mobile App**

1. Create a dummy Mobile app which will reflect the menu items available in each Qeu box which is similar to end customer accessing.
2. Provide a provision to change the location in the dummy mobile app, once changed the menu items available in that QeuBox location should be displayed
3. When dummy user selects a menu item from the dummy app and saves, this data should be sent to aggregator DB.
4. The Aggregator DB will send SKU to QeuBox {Order number key/QR code}
5. Block that menu item in aggregator app
6. A synchronized message will be sent to Qeu Box Instant communication API
7. Update the Server DB
8. Update the Remote DB with the Order number/QR code

* **Menu When picked up.**

1. Order number status will be marked completed/delivered
2. Updated status shared to Server
3. Updated status shared to aggregator.

* **QeuBOX reloaded – When a new menu item is reloaded into the QeuBOX compartment**

1. Update the QeuBOX compartment in remote
2. Update the QeuBOX DB in server
3. Publish the status in API – so that the same is reflected back to aggregator
   1. The Aggregator backend will save and then reflect the same into their app
4. These actions should be seamless and instantaneous.