A PROJECT REPORT ON

INTERNSHIP AT BOX8

A thesis submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

In Computer Science and Engineering

2018 -2019

By

Janmejay S Purohit

DSU15CS0027

Under the guidance of

Dr. Rajesh T.M.

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DAYANANDA SAGAR UNIVERSITY

Kudlu Gate, Hosur Road, Bangalore – 560068

DAYANANDA SAGAR UNIVERSITY

Kudlu Gate, Hosur Road, Bangalore – 560068

Department of Computer Science and Engineering

BONAFIDE CERTIFICATE

This is to certify that the internship project at Box8 is a bonafide record of the work done by Janmejay S Purohit (DSU15CS0027) in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering at the DAYANANDA SAGAR UNIVERSITY, BANGALORE, during the year 2018-2019.

Dr. Rajesh T.M.

Guide

External Examiner

Dr Bondu Venkat

Class Advisor

Dr. M K Banga

Chairman

Dr. A Srinivas

Dean - SoE

ABSTRACT

Payments in the backbone of any product-based company and for the customers to be happy, there should not be any discrepancies in the customer assets. It is the main motive of every company to have a structurally strong framework for payments and payment related proceedings. I have done my internship at Box8 in the payments team of the backend department. This report will tell you about the tasks I completed till date at Box8.

With emerging technologies and online payments, usage of various payment options has become a trend. Box8 wants to provide to its customers the liberty to use various payment options as available in the market. The duty bestowed to me was to integrate Google Pay payment gateway which uses the Omni Channel API as defined by the google developer docs.

With cases of failed transactions or transactions with canceled orders, refunds usually take 1-2 working days to reflect back in the customer's initial account used for payment mode. Since there was an absence of the knowledge about the situation between the initiation of the request and the completion, I implemented a refund status check which retrieves the information from the bank.

A worker is a set of code that runs by itself based on the configuration it was given. A worker was also made which runs by itself at a specified time to retrieve the status of a refund transaction and updates the same to the database. It runs by itself and does not need any manual interaction. Logging is done for reference at a point of time in the future.

Ongoing work right now is the Phone Pe in-App integration. In PhonePe app, there is a section called Apps. Here, I am having the backend integration part for Box8 to be a part of PhonePe apps in the food section of Apps in PhonePe. It is a work in progress and the final implementation will be done soon. Testing is in progress and the front end collaboration is in process.

ACKNOWLEDGEMENTS

I express my sincere thanks and gratitude to DAYANANDA SAGAR UNIVERSITY for providing me an opportunity to fulfill my most cherished desire of reaching my goal and thus helping me to make a bright career.

I am grateful to my project guide Dr. Rajesh T.M., Assistant Professor, Department of Computer Science and Engineering, DSU, Bangalore, for his valuable guidance, encouragement and for extending all possible help in timely completion of the project.

I would also like to express my sincere gratitude to my Class Advisor Dr Bondu Venkat, Associate Professor, Department of Computer Science & Engineering, DSU, Bangalore, for his constant support.

I am highly grateful to Dr. M.K Banga, Chairman, Department of Computer Science and Engineering, DSU, Bangalore, for his kind support, guidance and encouragement throughout the course of this internship work.

I express my heartfelt thanks to Dr. A Srinivas, Dean at the esteemed institution DSU Bangalore for providing with all the necessities to complete my internship.

I would like to thank all the teaching and non-teaching staff of Department of Computer Science and Engineering for their kind co-operation during the course of the internship work.

Finally, I am thankful to my parents and friends, who helped me in one way or the other throughout my project work.

Janmejay S Purohit

DSU15CS0027

DECLARATION

I, Janmejay S Purohit (DSU15CS0027), student of 8th semester B.Tech in Computer Science and Engineering, Dayananda Sagar University, Bengaluru, hereby declare that the internship project at Box8 submitted to the Dayananda Sagar University during the academic year 2018 - 2019, is a record of an original work done by me under the guidance of my manager Mr. Abhishek at Box8 and Dr. Rajesh T.M. assistant professor, Department of Computer Science and Engineering, Dayananda Sagar University, Bengaluru. This project work is submitted in partial fulfillment for the award of the degree of Bachelor of Technology. The result embodied has not been submitted to any other university or institute for the award of any degree.

Date: Janmejay S Purohit

Place: Bengaluru (DSU15CS0027)

TABLE OF CONTENTS

TABLE OF FIGURES

CHAPTER 1: INTRODUCTION

This report is a summary of my internship program at Box8 which was from 15th January 2019 to 15th June 2019 at Box8. My main duties included integration of payment gateways, bug fixes, server to server communication and payment transaction proceedings. During the course of the internship I worked on Ruby on Rails.

A payment gateway is a merchant service provided by any e-commerce application service provider for online payments. It authorizes credit card, debit card or direct payments processing for e-businesses, bricks and clicks, online retailers, or traditional brick and mortar. The payment gateway is given to the merchant by a bank, however, it can be provided by a specialized financial service provider as a separate service, such as a payment service provider.

A payment gateway helps make a payment transaction by the transfer of information between a payment portal (here, Box8) and the processor or the bank.

Other than integration of payment gateways, the tasks given to me were various bug fixes and server to server communication.

A refund status worker fetches the previous transactions batch-wise and logs the transactions and their refund state by making API calls to the bank server as listed in the payment gateway details of the details stored related to the transaction in our database. It also takes care of any promotions applied at the time of payment example given the Box8 money and updates the status in the refund details table.

Another segment was the refund checker, that assigns the status code to the refund transaction and determines its state whether the transaction is pending, approved or rejected by making server to server API calls.

There were many obstacles that came during the work, but with the help of my mentors I was able to solve them and deliver working modules.

CHAPTER 2: ABOUT BOX8

Box8 is a Mumbai-based on-demand food delivery company that managed to turn heads in a small amount of time. Founded by two IIT graduates Anshul Gupta & Amit Raj, BOX8 was started as a small outlet in a corporate cafeteria. As Box8 shares the quote “The idea was to serve Irresistible Desi Meals in a convenient, easy-to-carry box”.

Today, the company serves over 22,000+ meals every day across its 100+ outlets in Mumbai, Pune, Bangalore & Gurgaon!

It began as Poncho in 2011 serving just Mexican dishes in the quick service format, the company expanded its menu and rebranded (to Box8) in July 2012.

The co-founders Amit Raj and Anshul Gupta, both IIT alumni aimed to make their company the go to option for everyone who wanted to order great tasting food. Box8 is at present in Mumbai, Bangalore and Gurugram and has more than a hundred outlets over the cities. With more than 22,000 transactions for each day, Box8 has become 10x over the most recent years.

MOJO Pizza specializes in awesome Pizzas with presence at around 50+ different locations in Mumbai, Bangalore & Pune. The IT infrastructure is the same for both Box8 and MOJO Pizza. MOJO Pizza delivers delicious, wide range of pizzas.

Great care is taken to bifurcate the vegetarian and non-vegetarian racks at the kitchens and quality assessment is considered to be its top priority. Each outlet has an outlet manager who administers the total working and co-ordination of the outlet with respect to the delivery boys, customer care, order intake, refunds and complete administration.

Highly talented and motivated developers are hand picked from various IITs, NITs and top-notch universities across various cities of India and with the help of their ability, the technology at Box8 is top notch.

The outlets for both Box8 and MOJO Pizza are the same and are spread across Mumbai, Pune, Bangalore & Gurgaon.

The organization's strong attention towards taste, innovation and development has reverberated well with customers, as 80% of its day by day exchanges come from

rehash clients. Reasonable estimating, famous sustenance decisions and Indian cooking has reverberated well with its young urban client base.

Poncho Hospitality Private Limited is popularly known as Box8. The registered office of the company is at No.117, 27th Main, HRS Layout 2nd Sector, (Agara) Extension, Bangalore, Bangalore, Karnataka. MOJOPizza and Box8 are both brands under Poncho Hospitality Private Limited.

Below are some exciting facts about Box8:

- Rapid growth (10X) in last couple of years - selling 7 lakh+ meals per month.
- Well-Funded by ace venture funds like Mayfield, IIFL & IAN.
- Already profitable - poised to grow multi-fold in the coming quarters.
- Huge opportunity to emerge as a leader in this rapidly growing industry, a large part of which is still unorganized.
- Set to expand to 3 new cities over the next 6-9 months.
- The total paid-up capital is INR 11.77 lakhs.
- Box8 is open till 1 AM! Get piping hot meals delivered in under 38 mins with no delivery charges.
- Highly efficient frameworks and IT tools are implemented for seamless customer service
- 90+ delivery stores across 4 cities –Bangalore, Gurgaon, Mumbai & Pune.
- Top notch founding and leadership team from IITs/IIMs with previous work experience in top investment banks, consulting firms & product companies.

CHAPTER 3: JOB DESCRIPTION

⟶ Profile: Software Development Engineer Intern

⟶ Division: Backend

⟶ Team: Payments Integration Team

⟶ Responsibilities:

- ⟶ Monitor Payment Transactions
- ⟶ Integrate Payment Gateways
- ⟶ Bug Fixes
- ⟶ API Integrations
- ⟶ Testing for APIs

CHAPTER 4: SYSTEM REQUIREMENTS

The 4 requirements for full functionality of the environment is as under:

4.1. Functional Requirements:

- The database must be active and should fetch data seamlessly
- The third party servers should provide response data as defined in their documentation.
- Customer should be able to access Box8 and Mojo Pizza at all times.
- Latency should be handled.

4.2. Non functional Requirements:

- There should be a load balancer working when services are being deployed.
- The code should be easy to understand and should have readability.
- Graphical representation of data on Kibana must be configured.
- Redash queries should be dependable and reliable

4.3. Software Requirements:

- Frontend: HTML5, CSS, JavaScript, JQuery, AngularJS, Android Studio and Node
- Backend: Ruby on Rails, Rspec, PostgreSQL
- Server: nginx server using AWS
- Search Engine: Elasticsearch
- Cache control, Intermediate Storage and Queues: Redis

4.4. Hardware Requirements:

- RAM: 8GB
- Processor: Intel i5 or better
- Storage: 100GB or more

CHAPTER 5: LITERATURE REVIEW

Box8 API is the name given to the backend service used to build Box8 and MOJO Pizza client services.

Backend Services Environment

5.1. Ruby on Rails:

Box8 API is built on Ruby on Rails. Ruby on Rails is a server-side web application framework which is written in Ruby language. It follows a Model View Controller (MVC) Architecture. It is an easy to use framework and has inbuilt features for scaffolding default structures of a database, web service and web pages. Ruby on Rails is a widely popular framework with the least number of drawbacks, currently available for use.

Ruby on Rails emphasizes more on Convention over Configuration (CoC), Don't Repeat Yourself (DRY) and the infamous Active Record pattern.

Version used:

- Ruby: v2.4.2
- Rails: v5.0.4
- Bundler: v1.16.2

5.2. Rspec:

Box8 API undergoes thorough Integration Testing, Unit Testing, Black Box Testing, White Box Testing, User Acceptance Testing, Full stack testing and finally Sanity Testing.

RSpec is a Domain Specific Language (DSL) testing tool written in Ruby for the purpose of testing code written in Ruby. It is a behavior-driven development (BDD) framework which is widely used in production applications.

The key idea behind RSpec testing is that of Test Driven Development (TDD) where the tests are initially written and later on, the development of the code is done based on writing just enough code that will fulfill those tests followed by continuous code refactoring, reviews and deployments.

It has its own mocking framework that is completely integrated into the framework based upon a concept called JMock. The simplicity and ease of use in the RSpec syntax makes it one of the most popular and widely used testing tool for Ruby applications. It has 3 main blocks describe, context and it. RSpec is also having fixtures and factories which help create test data and seed test database during test run time.

Version used:

- Rspec: v3.7
  - Rspec-core: v3.7.0
  - Rspec-expectations: v3.7.0
  - Rspec-mocks: v3.7.0
  - Rspec-rails: v3.7.1
  - Rspec-support: v3.7.0

5.3. PostgreSQL:

The database on which the storage is done is Postgres, also known as PostgreSQL. It is a very powerful, free and open source relational database management system which boasts about the great care taken for extensibility, elasticity, feasibility and technical standards compliance. Postgres runs on all major operating systems and it is ACID compliant.

Version used: v10

5.4. Kibana:

Kibana is a data visualization tool which is used for various purposes. Data that is logged can be monitored, analyzed and based on the data, various predictions and recommendations are achieved. It is a plug in for elastic search for content that is indexed in an elastic cluster.

Elastic Stack or ELK is the combination of the tools used for data analytics and data engineering. Kibana has file beats and each file beat stores records related to the particular month.

Version: 7.1.1

5.5. Redis:

Redis is an open source (BSD licensed), in-memory data structure store, and is a key-value database with optional durability to store cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes with radius queries and streams.

Redis has built-in replication, Lua scripting, Least Recently Used eviction, transactions and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster.

Atomic operations can be run on these types, like appending to a string; incrementing the value in a hash; pushing an element to a list; computing set intersection, union and difference; or getting the member with highest ranking in a sorted set.

In order to achieve its outstanding performance, Redis works with an in-memory dataset. Depending on the use case, the dataset can be dumped to disk every once in a while, or each command can be logged. Persistence can be optionally disabled, if you just need a feature-rich, networked, in-memory cache.

Redis also supports trivial-to-setup master-slave asynchronous replication, with very fast non-blocking first synchronization, auto-reconnection with partial resynchronization on net split.

Other features include:

- Transactions
- Pub/Sub
- Lua scripting
- Keys with a limited time-to-live
- LRU eviction of keys
- Automatic failover

Redis is written in ANSI C and works in most POSIX systems like Linux, *BSD, OS X without external dependencies.

Version used:

- Redis-cli: v4.0.9

5.6. Redash:

Redash is an open source tool built for developers to query, visualize and collaborate data stored in their database. Redash is very quick to setup and works almost with any type of data source based on the requirement so as to query from anywhere in no time.

Once done, results can be shared along with the dashboards to the other team members and help the whole organization to be data driven with no-code filters and parameters that instantly adjust. Alerts can be sent for pre-defined triggers to email, Slack, or Hipchat (A custom webhook can be set up, as well).

Redash is our take on freeing the data within our company in a way that will better fit our culture and usage patterns. We tried to use traditional BI suites and discovered a set of bloated, technically challenged and slow tools/flows. What we were looking for was a more hacker'ish way to look at data, so we built one.

Python is integrated into redash which can help data processing by using elastic search. Python can be used for data mining and its applications. Alert systems can also be setup using python where a daily mailer is implemented that generates payment transaction success reports for the previous day's transactions sorted by the gateways.

Redash was built to achieve fast and easy access to billions and billions of records, that redash collects through Amazon Redshift ("petabyte scale data warehouse" that "speaks" PostgreSQL). As of recent times, Redash has support for querying multiple databases, including: Redshift, Google BigQuery, Google Spreadsheets, PostgreSQL, MySQL, Graphite, Axibase Time Series Database and custom scripts.

Version used:

- Redash: v7.0.0

5.7. GitLab:

Gitlab is a tool used for Version Control. It is a web based DevOps tool that provides a repository manager for project planning, collaboration, issue fixes, time management, micro coding and integration. GitLab is a single application for the entire software development lifecycle. From project planning and source code

management to CI/CD, monitoring, and security. GitLab enables teams to collaborate and work from a single conversation, instead of managing multiple threads across disparate tools. GitLab provides teams a single data store, one user interface, and one permission model across the DevOps lifecycle allowing teams to collaborate, significantly reducing cycle time and focus exclusively on building great software quickly.

Version used:

- Gitlab: v11.11

5.8. Sidekiq:

Sidekiq is a simple, open source, job schedule for background job processing in Ruby. Sidekiq does not do the scheduling, but it does the job processing. FIFO (First In First Out) methodology is implemented by Sidekiq. It takes in the job queue which is stored in redis and executes them. Sidekiq uses threads to handle multiple jobs at the same time occurring in the same process.

Scheduling can be added to sidekiq using cron. Cron is used to schedule commands at a specific time. These scheduled commands or tasks are known as “Cron Jobs”

Syntax of cron:

- Minute(0-59) Hour(0-24) Day_of_month(1-31) Month(1-12) Day_of_week(0-6) Command_to_execute

Example of cron:

- For 22:00 on every day-of-week from Monday through Friday
- 0 22 * * 1-5

Version used:

- V4.0.0

CHAPTER 6: REFUND STATUS CHECK

6.1. Objective:

The objective of implementing this is to enable a manual way to check for the status of a refund initiated by a customer by calling up the customer care of Box8/MOJOPizza. Sometimes the bank may take upto 2 or 3 days to complete a refund. To get an insight in the meanwhile, a refund status check is to be done.

6.2. Procedure:

The procedure follows a linear approach where the refund is mapped through the customer details which is acquired when the customer provides their phone number to the customer care executive.

1. check refund status button is clicked at POS
2. it routes to get_refund_staus function and sends refund_id as a parameter
3. The refund_id is used to identify the transaction
4. the transaction_status controller gets the refund details using get_refund_details function.
5. The details are received from the bank server
6. order_refund creates an object
7. Based on the gateway, a call is made to the respective check_refund for the gateway from refund_transaction_status module.
8. Once the response is received from the gateway, corresponding json is rendered.
9. If the gateway supports refund_check, status code is assigned as below.
    - Status code : 200 “approved”
    - Status code : 404 “rejected”
    - Status code : 422 “pending”
10. The same is updated in the database for the corresponding transaction
11. Details are updated in the POS and the customer gets the required information
12. RSpec Tests are written and all cases are tested.

6.3. Flowchart:

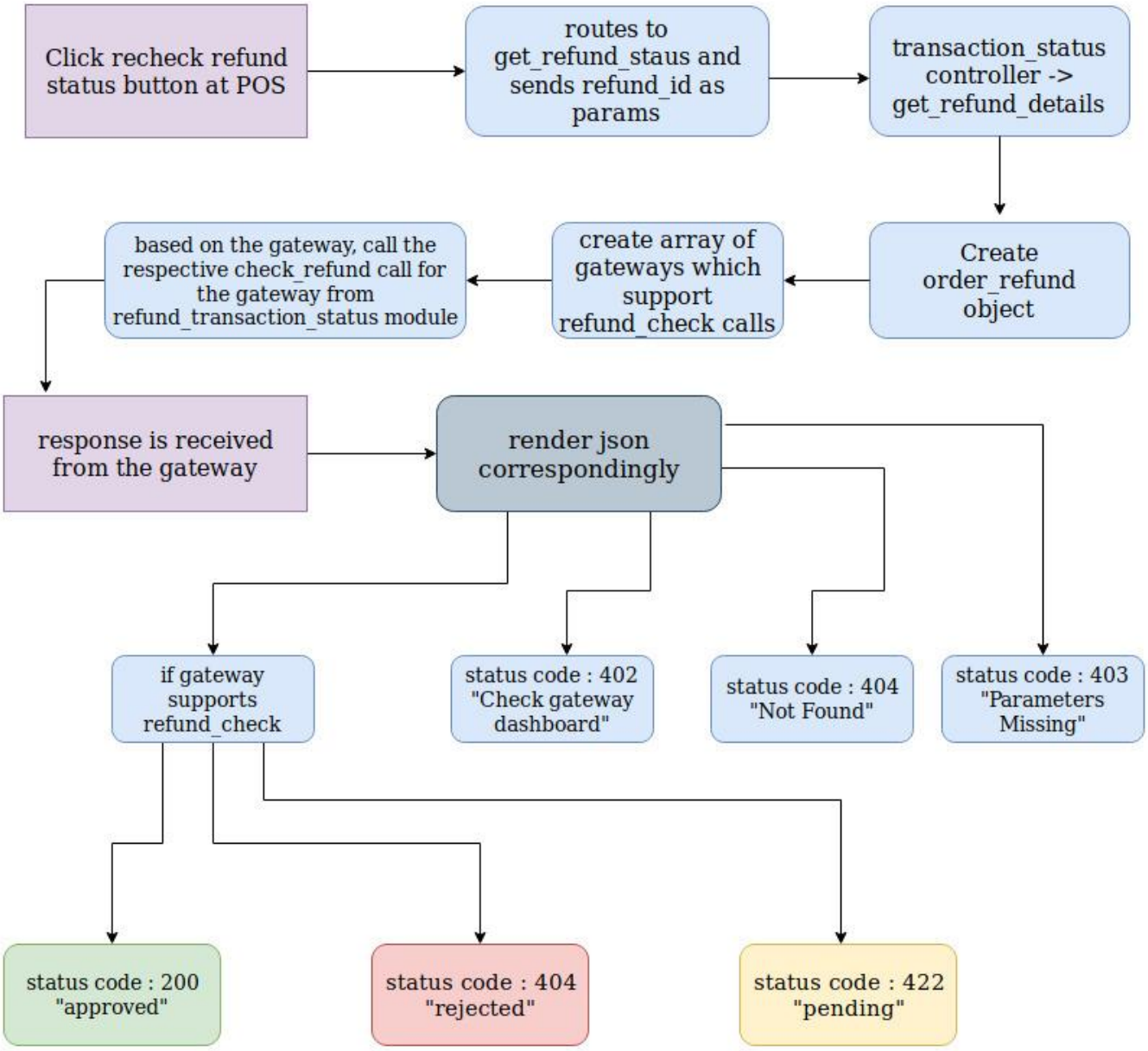

Fig 6.1: Refund Status Check Flowchart

6.4. Inputs:

- Customer's Phone Number: given by the customer
- Payment Transaction ID: fetched using the customer's phone number
- Refund ID: fetched using Payment Transaction ID

6.5. Outputs:

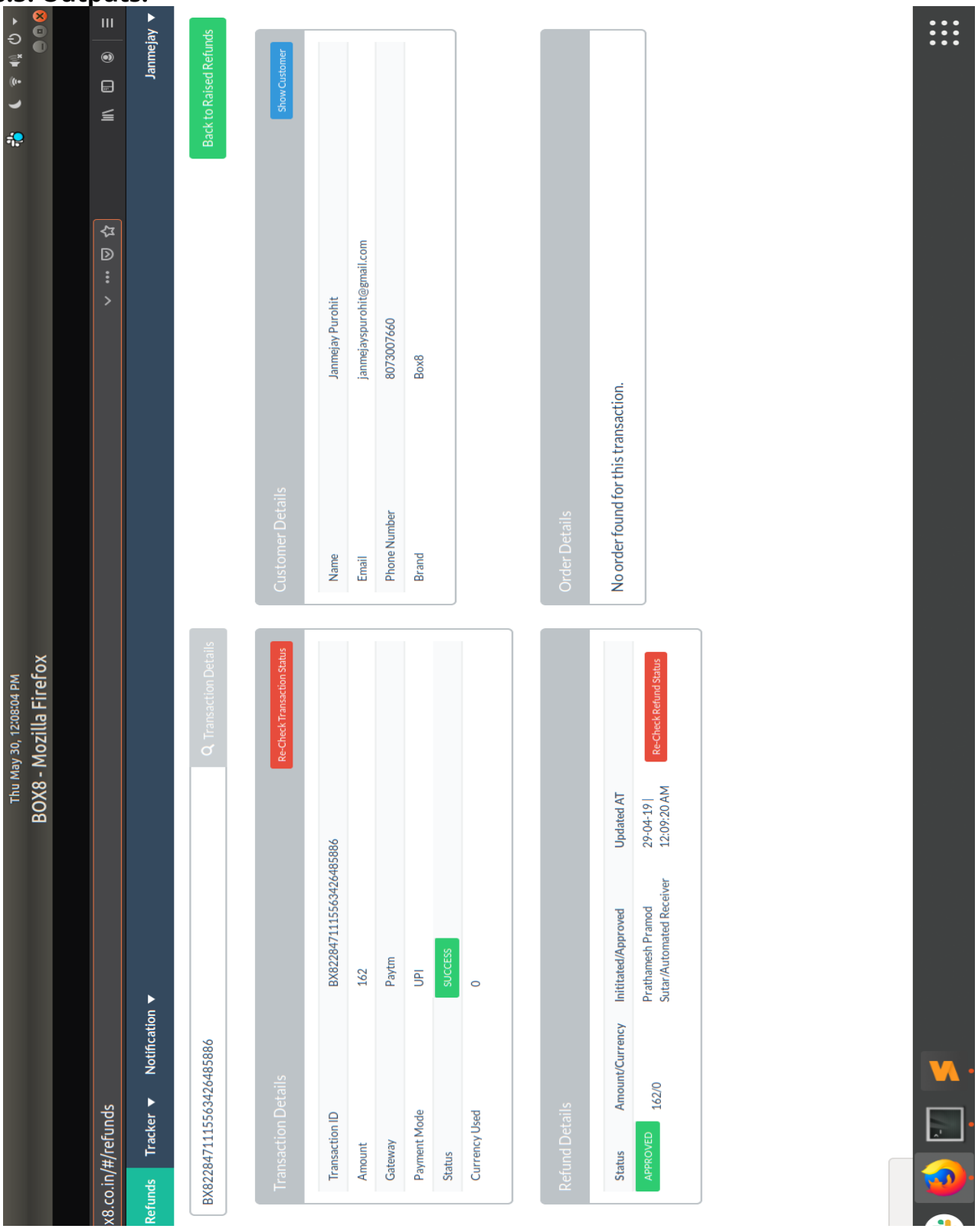

Fig 6.2: Refund Status Check Output

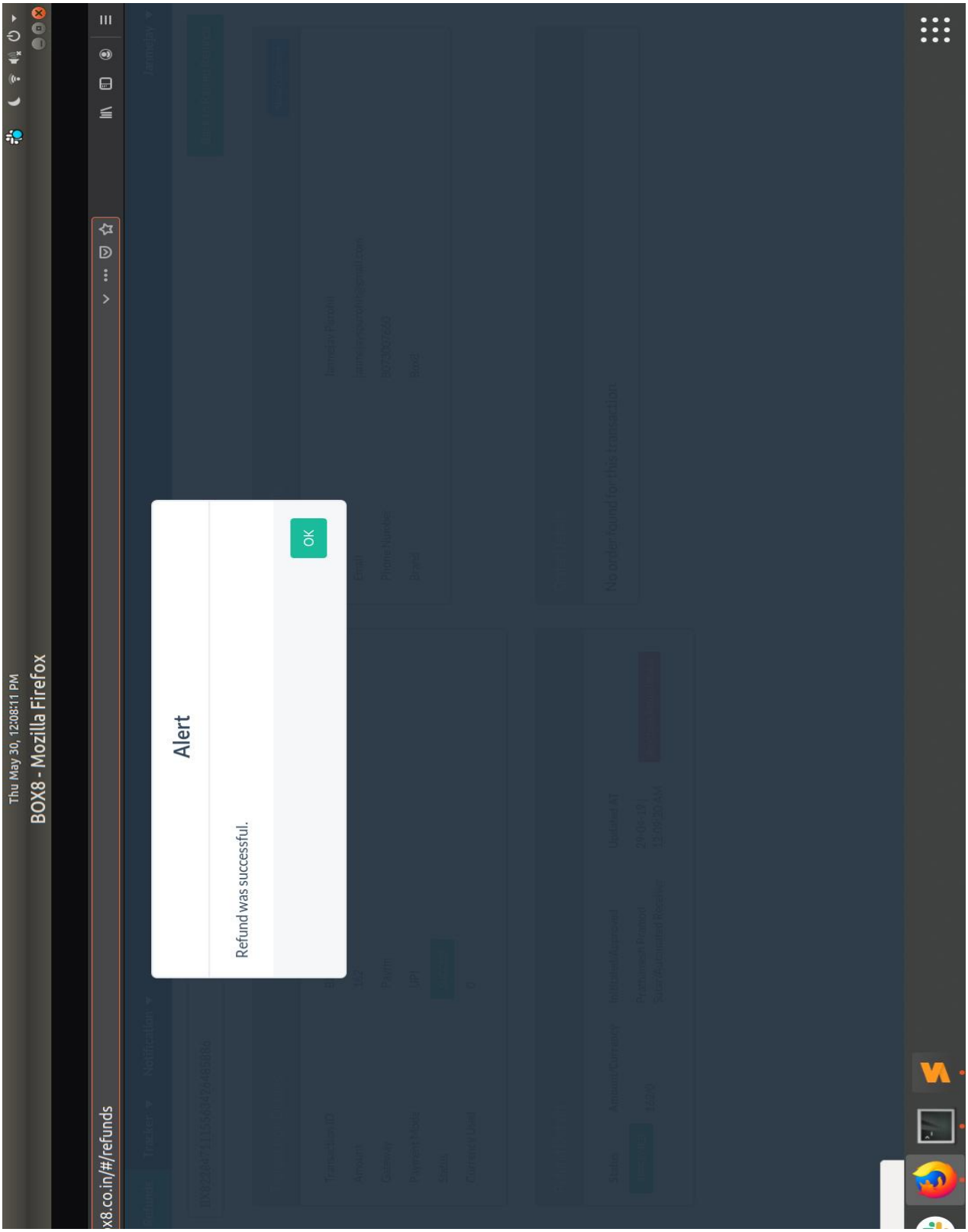

Fig 6.3: Refund Status Check Output

CHAPTER 7: REFUND INTEGRITY WORKER

7.1. Objective:

The objective of this worker is to automate things. As the worker runs asynchronously by itself, at the time specified by the cron job, no one has to manually run it. It is capable of handling refunds in bulk amount.

7.2. Procedure:

1. Every morning at 4 a.m. IST, the Sidekiq worker starts its execution.
2. Its first job is to determine and fetch all the pending transactions from the previous day to 3 days ago.
3. Once fetched, the transactions are executed in a batch of hundred until exhausted.
4. The next step is to obtain and select the refund from current batch. Further processes are repeated for each refund transaction.
5. Once the refund transaction is selected from the batch, the next task is to get the corresponding payment transaction for the refund.
6. The corresponding gateway name and Refund ID is logged.
7. In the next step, we make a call to the gateway.
8. Next the extraction of the refund status is done from the gateway by Faraday HTTP Request.
9. After the extraction of the refund status, it is categorized based on the response from the payload into the following four categories, namely
    - Pending (step 10)
    - Failed (step 13)
    - Gateway Timeout (step 14)
    - Success(step 15)
10. In case of a pending refund status, the refund transaction and worker state details are logged.
11. Then it is checked if the return request is older than two days, if so job is enqueues to emailer_queue and transaction details are sent to the tech team as an email.
12. After that the refund id and state is added to the refund_statuses array.
13. In case of failed transaction status, the refund status is updated to failed and refund id and state is added to the refund_statuses array.

14. In case of gateway time-out refund status, the refund status is updated to gateway_timeout and refund id and state is added to the refund_statuses array.
15. In case of success transaction status, the refund status is updated to success and checked if the payment has box8 wallet money.
16. If yes, then a refund is created for the corresponding amount of box8 wallet money and the respective refund id and state is added to the refund_statuses array else the refund id and state is directly added to the refund_statuses array.
17. After the updation in the refund_statuses array the next refund transaction is selected and the same steps are followed until all batches are exhausted.
18. RSpec Tests are written and all cases are tested.

7.3. Input:

All inputs are automatically fetched by querying to the database for refund transactions whose status is pending.

7.4. Flowchart:

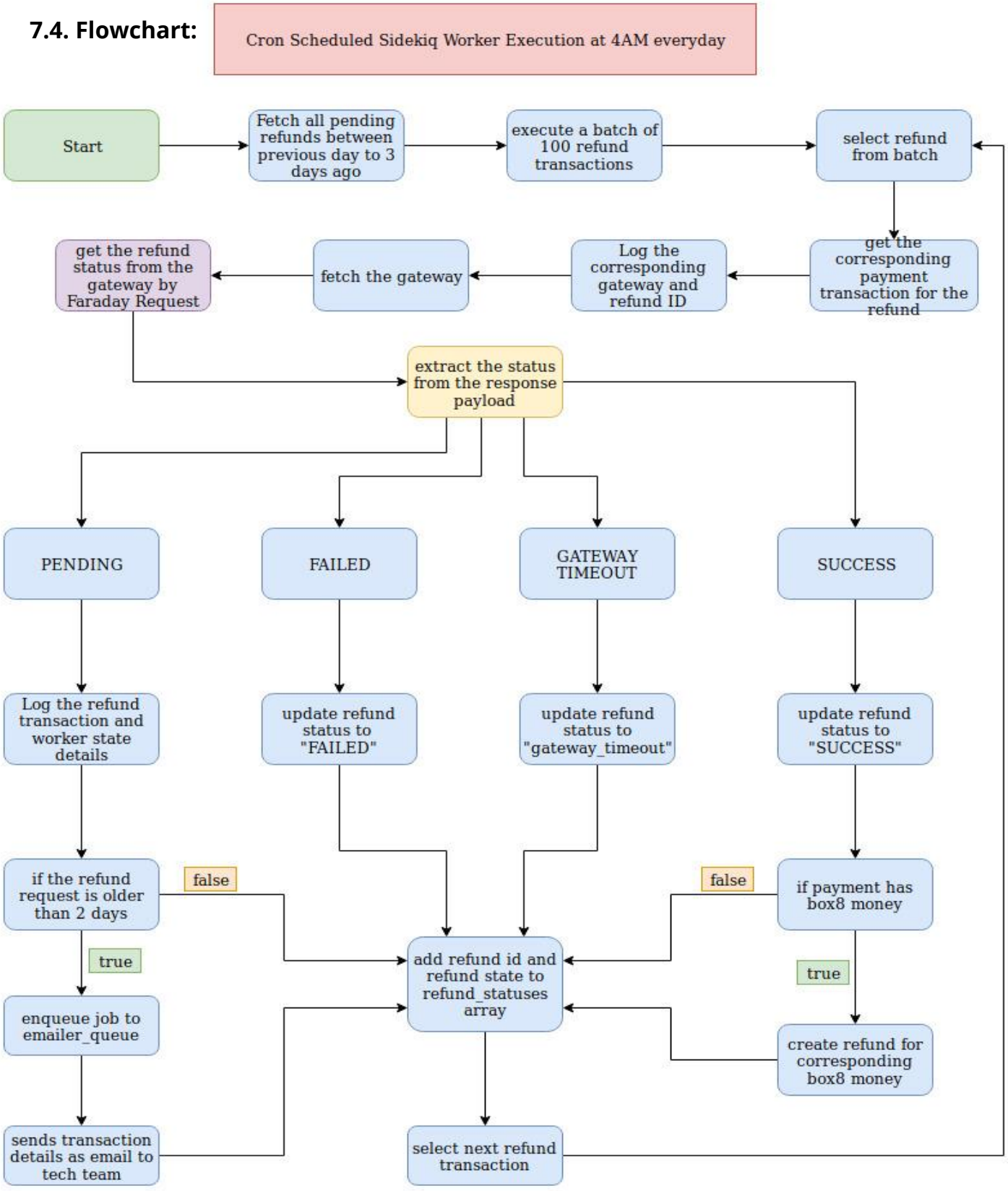

Fig 7.1: REFUND INTEGRITY WORKER FLOWCHART

7.5. Output:

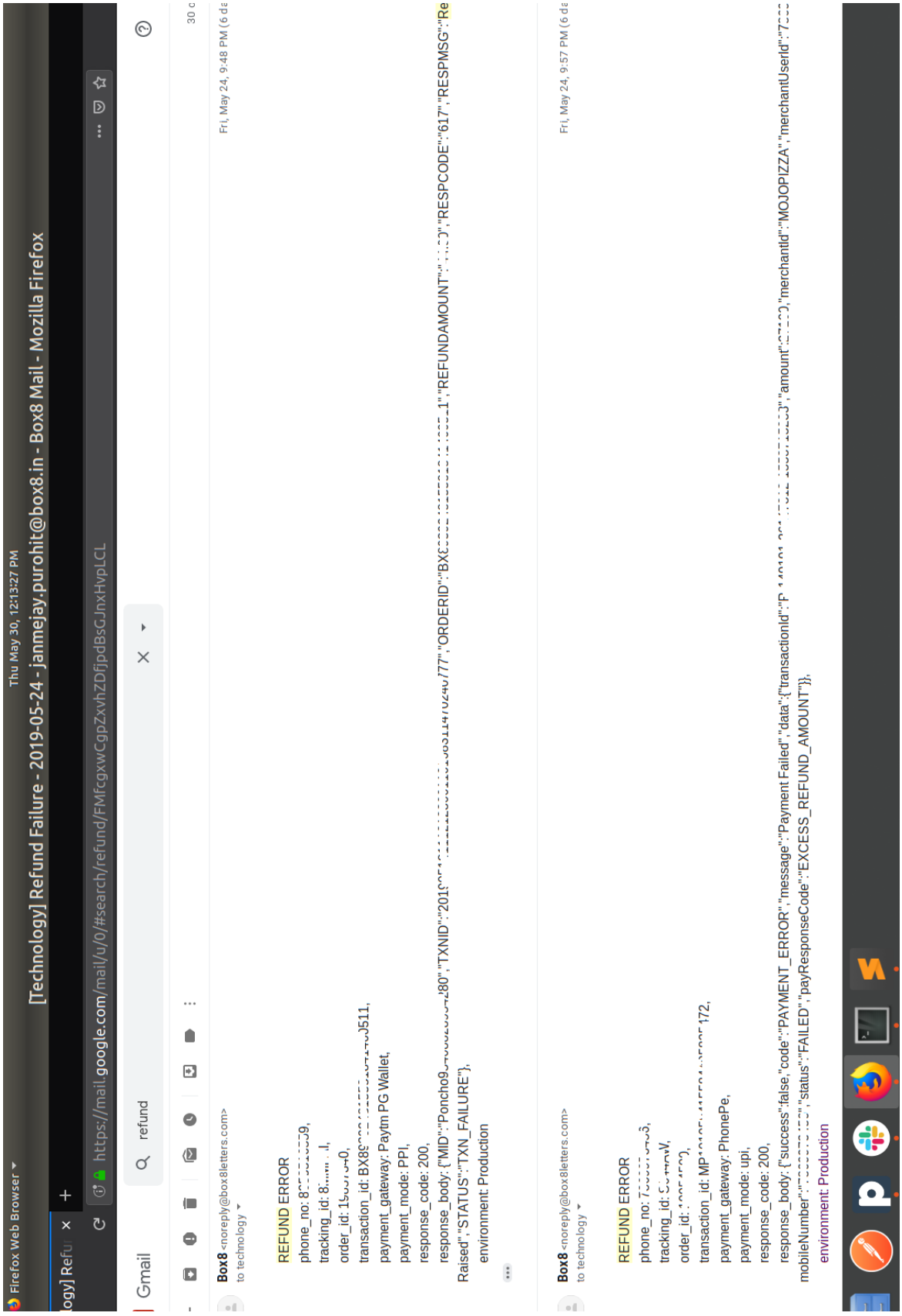

Fig 7.2

CHAPTER 8: GOOGLE PAY INTEGRATION

8.1. Objective:

The objective of implementing Google Pay is to provide customers to use Google Pay for making their payments for their Box8 and MOJOPizza orders. Google Pay is a convenient and highly used payment platform so this will help customers to checkout their orders easily.

8.2. APIs:

The following list consist of the APIs implemented in the Google Pay controller:

1. inititate_collect
2. callback
3. check_status
4. initiate_intent

8.3. Process (along with inputs and response):

The flow consists of 2 phases :

1. **Initiate phase**
2. **Validate phase**

Step 1: Initiate Phase:

i) Initiate collect phase :

- Customer clicks ‘pay using Google Pay’ button.
- Pop-up shown with pre-populated Customer’s mobile number with option to edit.
- Payment transaction is created for the Customer.
- Transaction details are generated and stored.
- OAuth Credential is generated for the current transaction.
- Payload sent to Google Pay API’s link, as provided and instructed by Google
- Status response is received and appropriate json is rendered.

```
API : initiate_collect
Method : POST
Params : {
            header: session-id (string),
            body:
                  'currency_redeem' (boolean),
                  'payment_option_id' (integer),
                  'address_id' (integer),
                  'remark' (string),
                  'outlet_service_type' (string),
                  'order_time' (long integer),
                  'cash_order_id' (integer),
                  'client_phone_number' (string)
          }
URI : /gpay/initiate_collect
```

- **Possible outcomes** : The response contains any of the following possible

values for the transactionStatus object.

- 200 : All OK
- 400 : Request contains invalid Phone Number or wrong parameters
- 404 : Account linked to Phone number was not found
- 409: Duplicate transaction

Sample success response

```
{
    "Data":{
       "merchantTxnId": " BX81088888888888857451",
       "Time_to_live":"185"
     },
    "meta":{
        “code”: 200 ,
        “status”: " OK" ,
        “message”: "All OK!" ,
        “error”: false ,
        “version”: "1.0" ,
        "copyright": "Copyright 2015 Box8"
    }
}
```

Sample invalid phone number response


```
{
        "data": {
        "merchantTxnId": " BX81088888888888857451",
        "time_to_live": "185"
        },
        "meta": {
        "code": 400,
        "status": "Not OK",
        "message": "Request contains invalid Phone Number or wrong parameters",
        "error": true,
        "version": "1.0",
        "copyright": "Copyright 2015 Box8"
        }
}
```


Sample account not found response


```
{
        "data": {
        "merchantTxnId": " BX81088888888888857451",
        "time_to_live": "185"
        },
        "meta": {
        "code": 400,
        "status": "Not OK",
        "message": "Account linked to Phone number was not found",
        "error": true,
        "version": "1.0",
        "copyright": "Copyright 2015 Box8"
        }
}
```

Sample duplicate transaction response

```
{
    "Data":{
       "merchantTxnId": "BX810656811551879858617 9",
       "Time_to_live":"185"
     },
    "meta":{
        “code”:  4009 ,
        “status”:  "Not OK" ,
        “message”:  "Duplicate Transaction" ,
        “error”:  true ,
        “version”:  "1.0" ,
        "copyright":  "Copyright 2015 Box8"
    }
}
```

ii). Initiate intent phase :

- Check for device compatibility (isReadyToPay) & whether the device is
- android only
- If both the response is true, initiate using intent flow, else collect flow

```
API : initiate_intent
Method : POST
Params : {
            header: session-id (string),
            body:
                'currency_redeem' (boolean),
                'payment_option_id' (integer),
                'address_id' (integer),
                'remark' (string),
                'outlet_service_type' (string),
                'order_time' (long integer),
                'cash_order_id' (integer),
        }
URI : /gpay/initiate_intent
```

Sample response for initiate intent

```
{
    "data": {
        pay_details = {
         "apiVersion" : 2,
         "apiVersionMinor" : 0,
         "allowedPaymentMethods" : [
         {
         "type" : "UPI",
         "params" : {
         "payeeVpa": box8vpa @axisbank ,
         "payeeName": Box8,
         "mcc": "5812" ,
         "transactionReferenceId": BX81088888888888857451,                    }
         "tokenizationSpecification" : {
         "type" : "DIRECT"
         }
         }],
         "transactionInfo" : {
         "totalPriceStatus" : "FINAL",
         "totalPrice" : "100",
         "currencyCode" : "INR",
         "transactionNote" : "Payment for your Box8 order
                  with Transaction ID BX81088888888888857451",
         }
         }
    },
    "meta": {
        “code”:  200 ,
        “status”:  "OK" ,
        “message”:  "Intent Pay Params Generated" ,
        “error”:  false ,
        “version”:  "1.0" ,
        "copyright":  "Copyright 2015 Box8"
    }
}
```

Sample not android transaction response

```
{
    "data": [],
    "meta": {
        “code”: 400 ,
        “status”: "Not OK" ,
        “message”: "Try Collect flow" ,
        “error”: true ,
        “version”: "1.0" ,
        "copyright": "Copyright 2015 Box8"
    }
}
```

Step 2: Validate phase:

- Polling starts after 30s of Google API call, with an interval of 5s between each request in case where there no callback received.

Collect Flow

```
API : check_status
Method : POST
Params : { header: session-id ( string ),
           body: ‘merchantTxnId’(string)
         }
URI : /gpay/check_status
```

Intent Flow

```
API : check_status
Method : POST
Params : { header: session-id ( string ),
           body: ‘merchantTxnId’(string),
                 ‘flow_type’ (string),
                 ‘reference_id’ (string)
         }
URI : /gpay/check_status
```

Possible outcomes : The response contains any of the following possible values for the transactionStatus object.

- SUCCESS
- FAILURE: Transaction has failed
- IN_PROGRESS: Transaction is in progress
- PAYMENT_NOT_INITIATED: Payment hasn't been initiated by the user
- DECLINED: Payment has been declined by the user
- EXPIRED: Payment request expired
- DOES NOT EXIST

If box8 server got the callback from axis server, it checks the status of the payment transaction from database and sends the response to box8 client. Otherwise it makes another api call to axis server for checking status of transaction and then responds back to t he client.

Sample Successful transaction response


```
{
        "data": {
        "error": false,
        "merchantTxnId": " BX81088888888888857451",
        "amount": "4.0",
        "msg": "Payment Successful. Order Created.",
        "status": "SUCCESS",
        "transaction_successful": true
        },
        "callback_success": true,
        "meta": {
        "code": 200,
        "status": "OK",
        "message": "Payment Successful. Order Created.",
        "error": false,
        "version": "1.0",
        "copyright": "Copyright 2015 Box8"
        }
}
```

Sample Payment Declined response


```
{
        "data": {
        "error": true,
        "merchantTxnId": " BX81088888888888857451",
        "amount": 99,
        "msg": "Payment failure",
        "status": "DECLINED",
        "transaction_successful": false
        },
        "callback_success": true,
        "meta": {
        "code": 400,
        "status": "Not OK",
        "message": "Payment failed",
        "error": true,
        "version": "1.0",
        "copyright": "Copyright 2015 Box8"
        }
}
```


Sample Payment not found response


```
{
        "data": {
        "error": true,
        "merchantTxnId": " BX81088888888888857451",
        "amount": 0,
        "msg": "Payment failure",
        "status": "NOT_FOUND",
        "transaction_successful": false
        },
        "callback_success": true,
        "meta": {
        "code": 400,
        "status": "Not OK",
        "message": "Payment failed",
        "error": true,
        "version": "1.0",
        "copyright": "Copyright 2015 Box8"
        }
}
```

Sample Payment pending response


```
{
        "data": [],
        "callback_success": false,
        "meta": {
        "code": 400,
        "status": "Not OK",
        "message": "Payment pending",
        "error": true,
        "version": "1.0",
        "copyright": "Copyright 2015 Box8"
        }
}
```


Sample expired state response


```
{
        "data": {
        "error": true,
        "merchantTxnId": " BX810888888888857451",
        "amount": 99,
        "msg": "Payment failure",
        "status": "EXPIRED",
        "transaction_successful": false
        },
        "callback_success": true,
        "meta": {
        "code": 400,
        "status": "Not OK",
        "message": "Payment failed",
        "error": true,
        "version": "1.0",
        "copyright": "Copyright 2015 Box8"
        }
}
```

Sample transaction does not exist response


```
{
        "data": {
        "error": true,
        "msg": "Transaction does not exist",
        "transaction_successful": false
        },
        "meta": {
        "code": 409,
        "status": "Not OK",
        "message": "Transaction does not exist",
        "error": true,
        "body": null,
        "version": "1.0",
        "copyright": "Copyright 2015 Box8"
        }
}
```


Sample for any other (invalid) response


```
{
        "data": [],
        "callback_success": true,
        "meta": {
        "code": 409,
        "status": "Not OK",
        "message": "Unknown State.",
        "error": true,
        "body": null,
        "version": "1.0",
        "copyright": "Copyright 2015 Box8"
        }
}
```

- Axis server sends a callback to box8 server regarding the status of transaction.
- Order is created on the basis of callback also. Basically, it's a race condition between check status api and callback api, whichever gets the information about transaction success state first, makes the order.
- Axis sends aes encrypted payload for every transaction on box8 server. Payload is decrypted with aes key provided by axis and the checksum is matched with rsa private key (public key is provided to axis by us for checksum encryption). Order is created (if not already) on the basis of payment status.
- Box8 client gets the payment status and it can either continue to poll (payment status pending) or redirect to cart (in case of web) / payment page(in case of app) for failed payment or stop polling (transaction success). Order will be created in transaction success case.
- RSpec Tests are written and all cases are tested.

Response in order successfully created


```
{
    "data":{
        "error": false ,
        "merchantTxnId": "BX810655501535357675 7451" ,
        "amount": "1.0" ,
        "msg": "Payment Successful. Order Created." ,
        "status": "SUCCESS" ,
    },
    "transaction_successful": true
    "meta":{
        "code": 200 ,
        "status": "OK" ,
        "message": "Payment Successful. Order Created." ,
        "error": false ,
        "version": "1.0" ,
        "copyright": "Copyright 2015 Box8"
    }
}
```

Response in case of payment success but item gets sold out after payment


```
{
    "data":{
        "error": true,
        "merchantTxnId": "BX8106555015353576757451",
        "amount": "1.0",
        "msg": "Payment Successful, order couldn't be processed",
        "status": "FAILED",
        "transaction_successful": true
    },
    "meta":{
        "code": 200,
        "status": "OK",
        "message": "Payment Successful, order couldn't be processed",
        "error": false,
        "version": "1.0",
        "copyright": "Copyright 2015 Box8"
    }
}
```


Response in amount mismatch


```
{
    "data":{
        "error": true,
        "merchantTxnId": "BX8106555015353576757451",
        "amount": "1.0",
        "msg": "Payment Successful, order couldn't be processed",
        "status": "FAILED",
        "transaction_successful": true
    },
    "meta":{
        "code": 200,
        "status": "OK",
        "message": "Payment Successful, order couldn't be processed",
        "error": false,
        "version": "1.0",
        "copyright": "Copyright 2015 Box8"
    }
}
```

8.4. Flowchart:

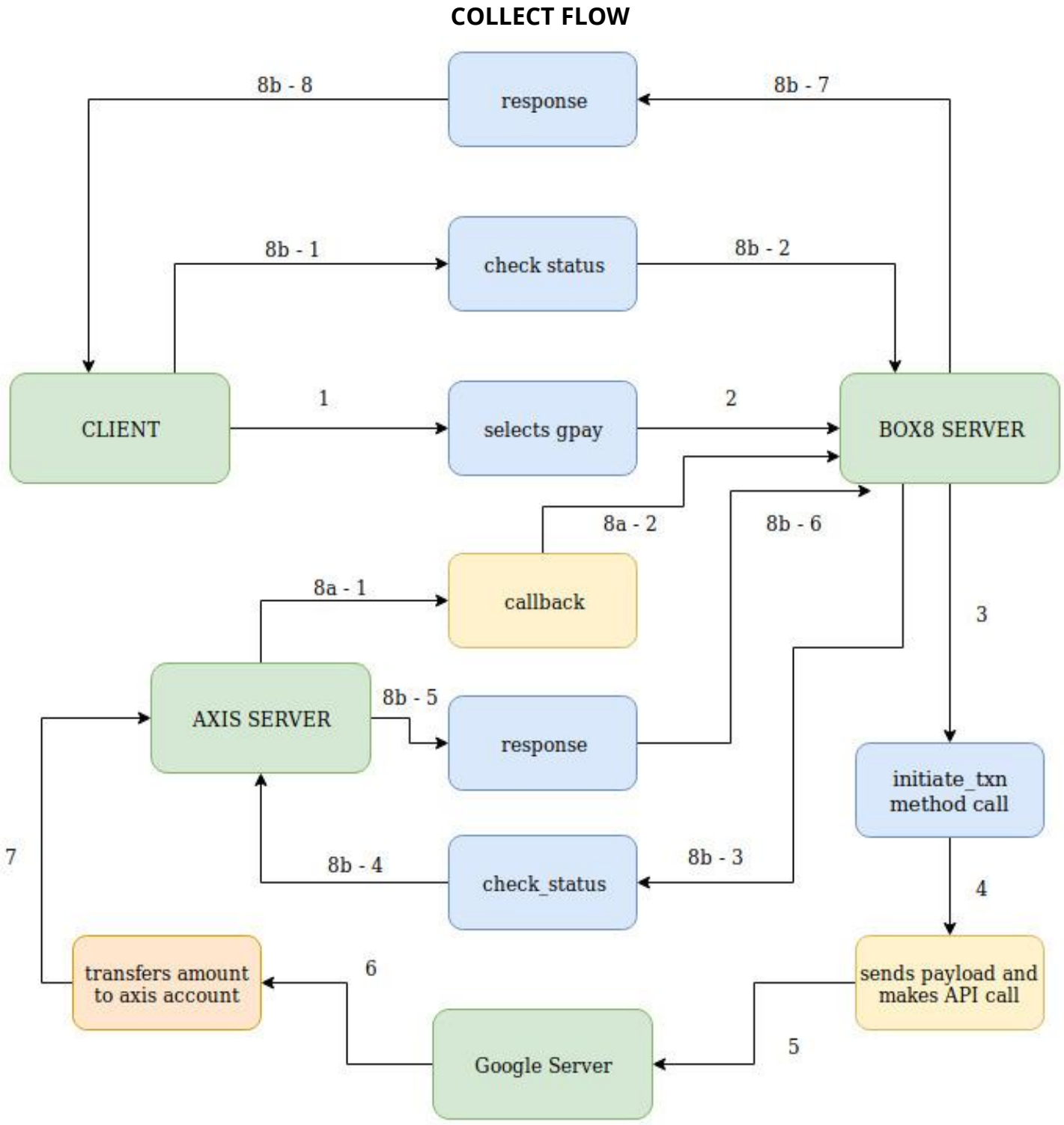

Fig 8.1: Google Pay Collect Flowchart

INTENT FLOW

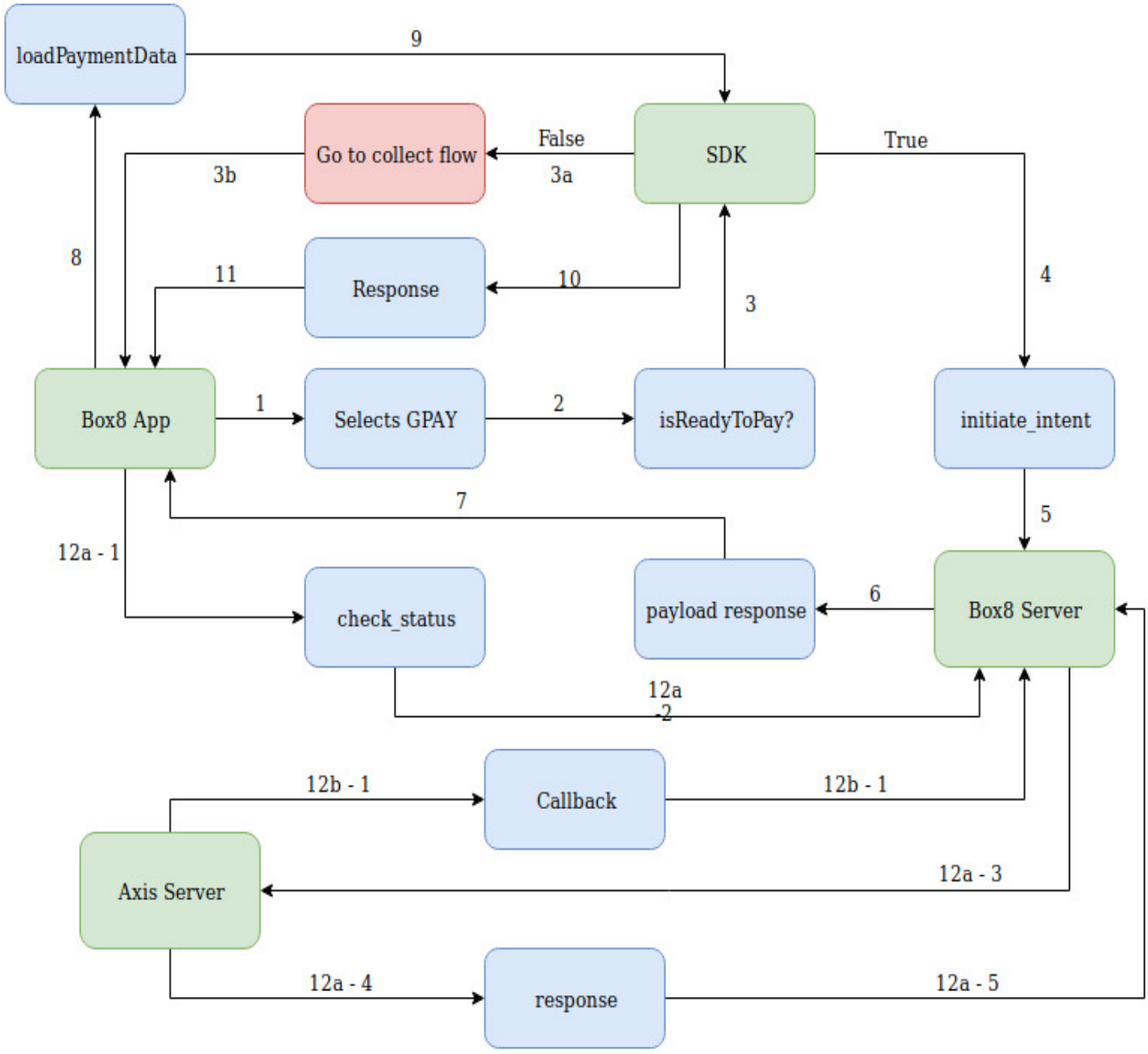

Fig 8.2: Google Pay Intent Flowchart

8.5. Output:

INTENT FLOW

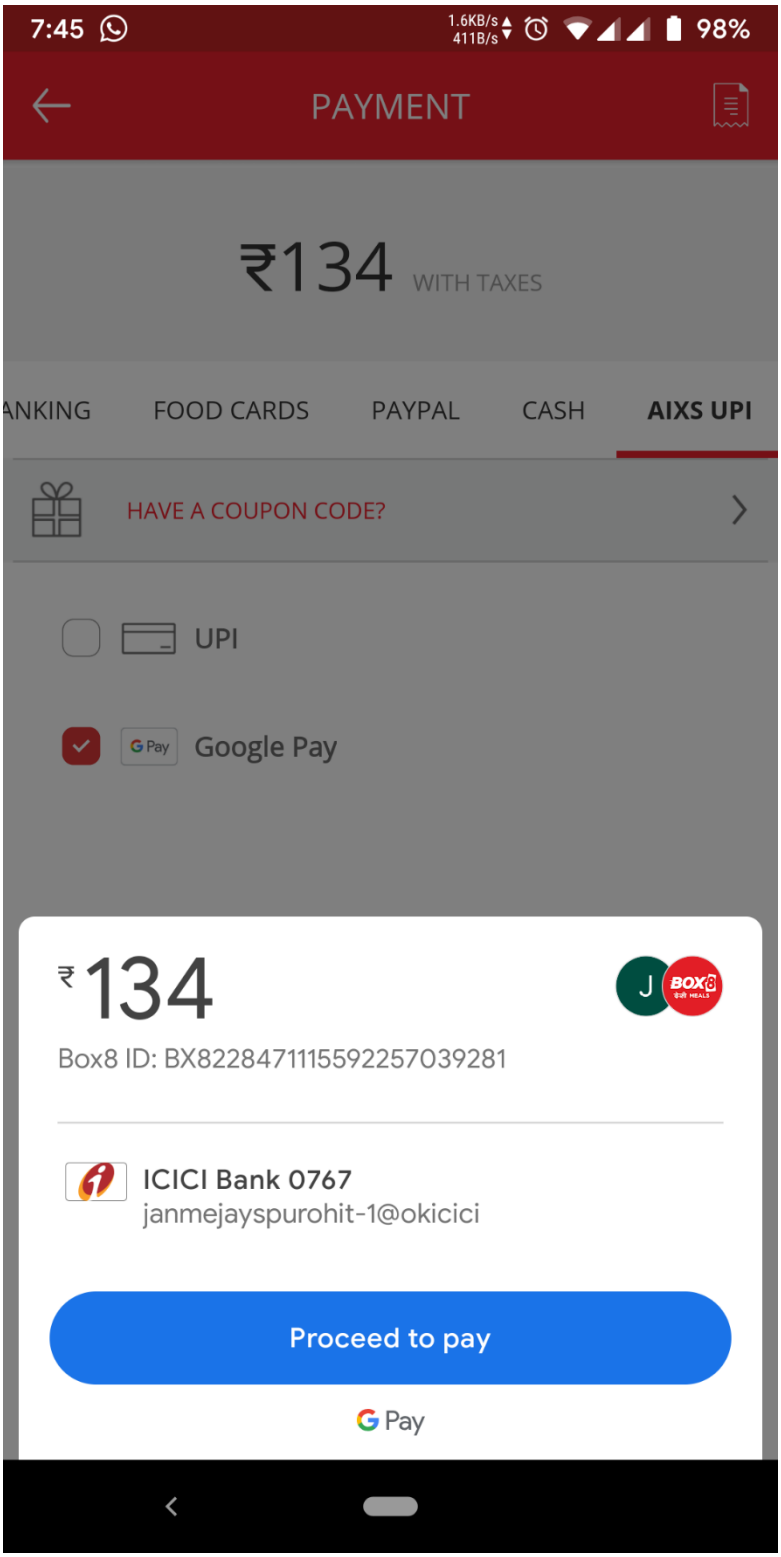

Fig 8.3

COLLECT FLOW

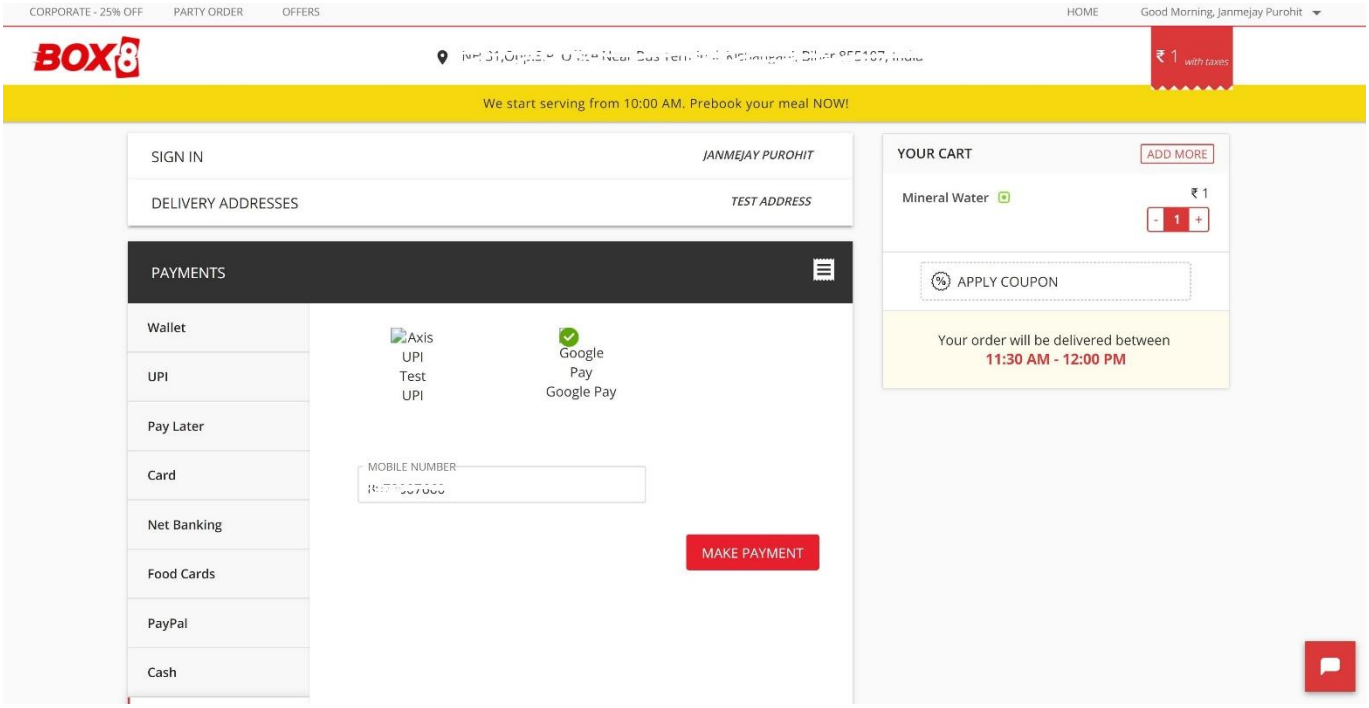

Fig 8.4

Fig 8.5

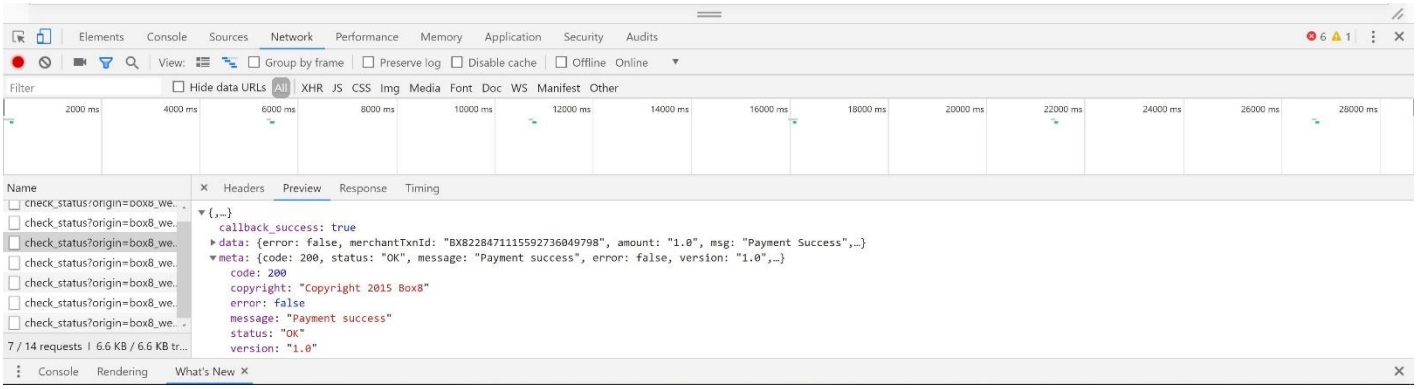

Fig 8.6 Polling

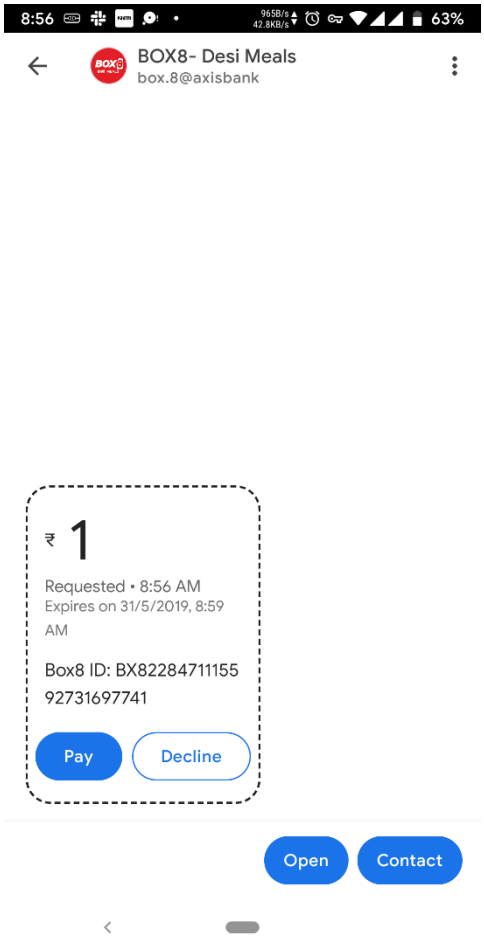

Fig 8.7

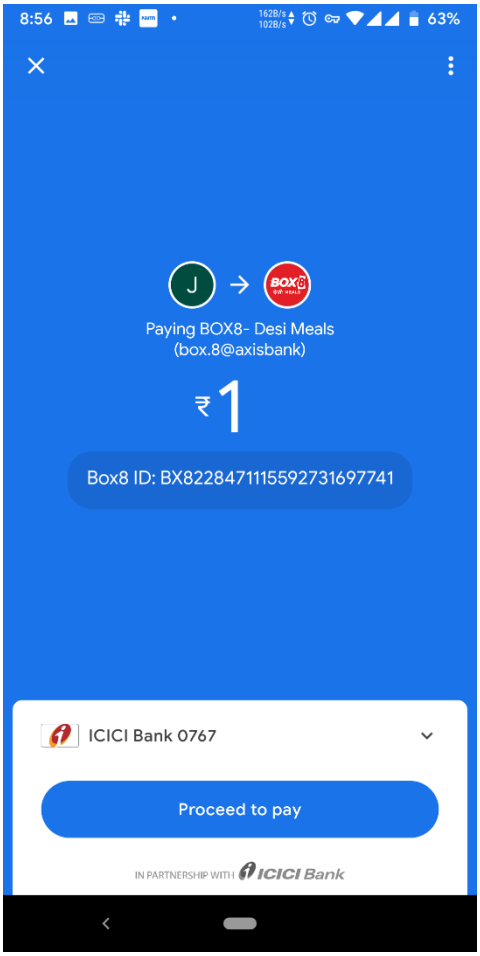

Fig 8.8

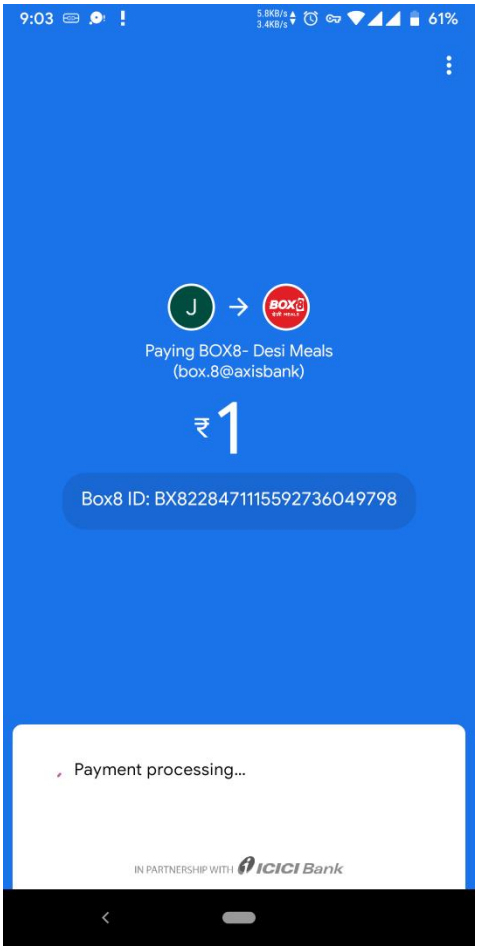

Fig 8.9

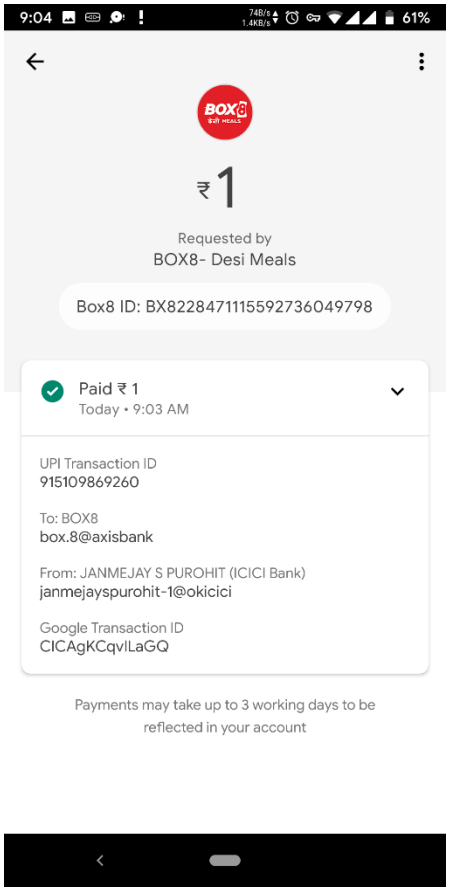

Fig 8.10

Fig 8.11

Order Summary

Tracking Id : YZYMZX

Name : Janmejay Purohit

Contact Number : 8073007660

Delivery Address : Take away - Test Address

Total Payable: ₹ 1.0

Mode: Online Paid

Order Details

Name of the item	Quantity
Mineral Water -	1

For immediate (60 mins) consumption only.

CHAPTER 9: PHONEPE INAPP INTEGRATION

9.1. Objective:

InApp is a Merchant's Progressive Web Application present inside PhonePe's "Apps" section. The objective of this task is to include Box8 into the InApps section of PhonePe to have its presence and thus compete in the food section of the PhonePe eco system.

9.2. Procedure:

1. Register Box8 with PhonePe and receive the merchant credentials
2. Install PhonePe JavaScript SDK (JS-SDK) and create PhonePe instance for the same
3. Process begins when customer selects Box8 from PhonePe InApps
4. JS-SDK invokes Box8 Web App
5. JS-SDK requests customer to get access to the location
6. If customer agrees then location is used to fetch the outlet, else a default outlet is shown and a manual input for the location is expected.
7. The customer then builds the cart
8. Once the cart is built, the JS-SDK asks the permission to share the customer's details with Box8.
9. If the customer denies to share then a manual login is awaited.
10. If customer agrees then a grant token is generated and sent to Box8 Server
11. Box8 server makes an API call to the PhonePe server using the grant token to fetch an SSO (Single Sign On) Token.

    The result may be a:

    - Success
    - Timeout
    - Wrong Credentials
    - Bad Request
12. All the server to server communication are SHA256 encrypted and Base64 encoded.
13. The SSO Token consists of customer's name and email id in encrypted form
14. The details are decrypted.

15. If the customer is an existing customer, the customer is authenticated and logged in.
16. If the customer is a new customer, an account is created for the customer.
17. The unsigned session id then gets signed and the cart is reassigned with the new session id
18. The customer then needs to make payment. JS-SDK is invoked and it redirects to the PhonePe payments page.
19. Once the payment is made, JS-SDK redirects back to Box8 and validation takes place
20. The status of the transaction is fetched and it could be in either of the states:
    - Success
    - Pending
    - Failed
    - Gateway Error
21. Based on the response, appropriate redirection is done.
22. A webhook is also received for these transactions.
23. The webhook is to be decrypted and is helpful where the validation results in a pending state but the webhook gives a success case.
24. To prevent creation of multiple orders for a single transaction, race condition check is made.
25. A key is generated and stored in redis using which optimistic locking approach is implemented.
26. Either the validate call or the webhook call can proceed to making an order, and not both
27. The order once created, locks the key for that transaction.

9.3. Flowchart:

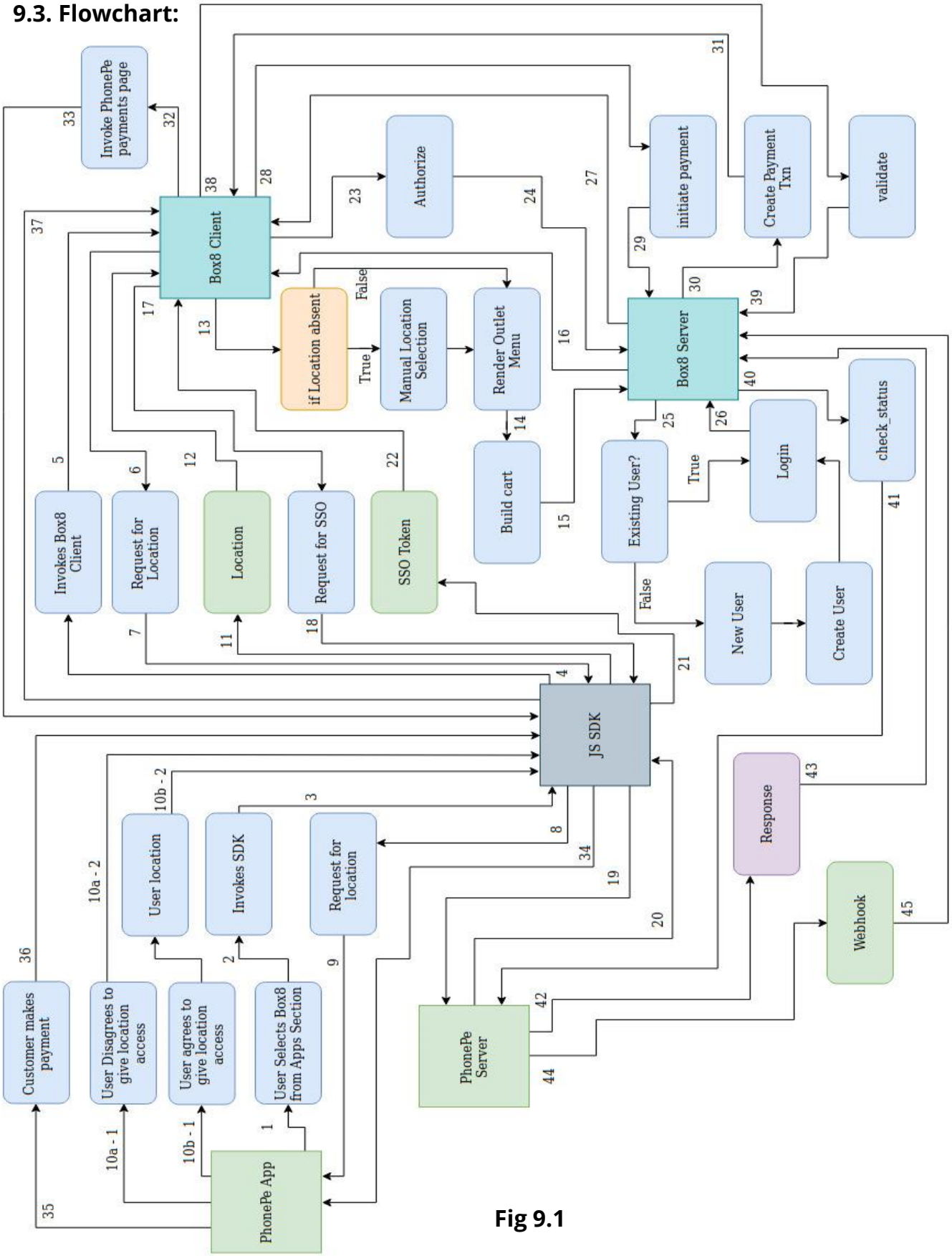

Fig 9.1

9.4. Inputs:

- Customer's location
- Customer's Name
- Customer's Phone Number
- Order Details
- Delivery Time
- Merchant ID
- Salt
- JS-SDK

9.5. Outputs:

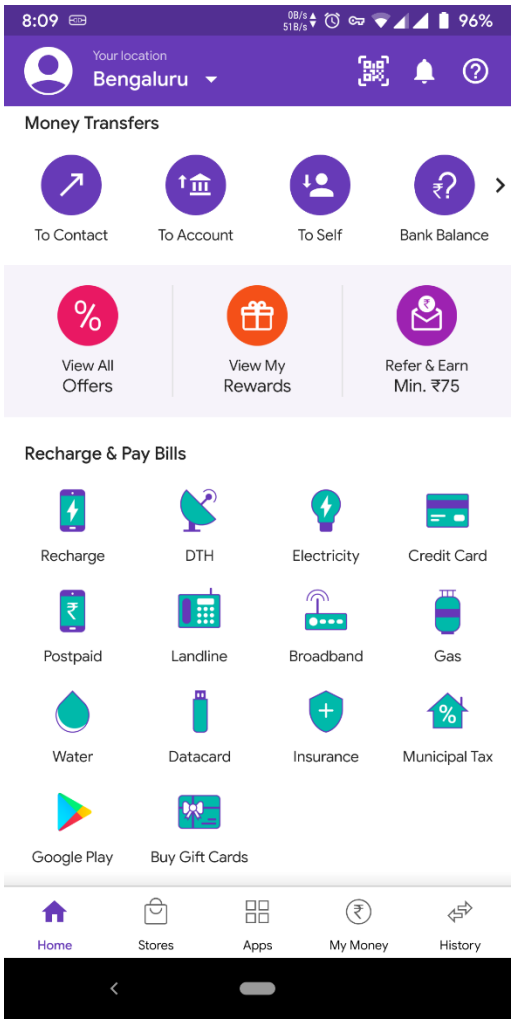

Fig 9.2

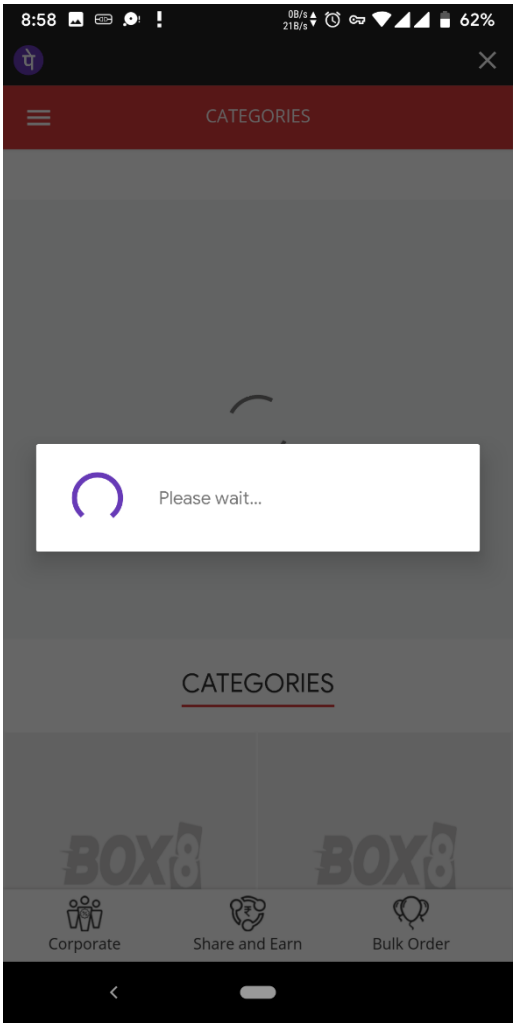

Fig 9.3

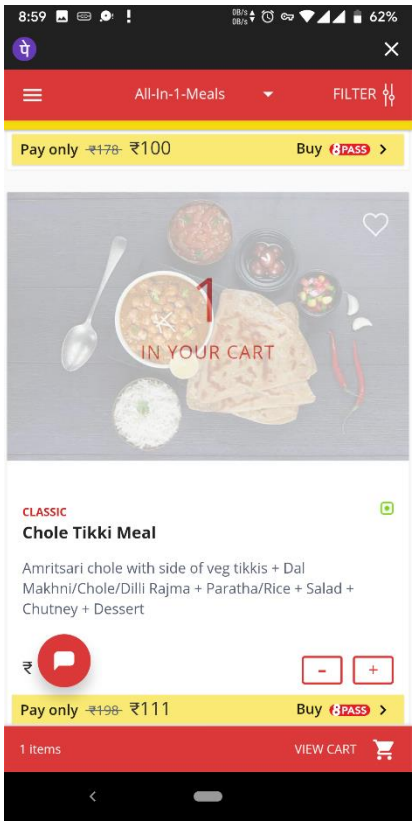

Fig 9.4

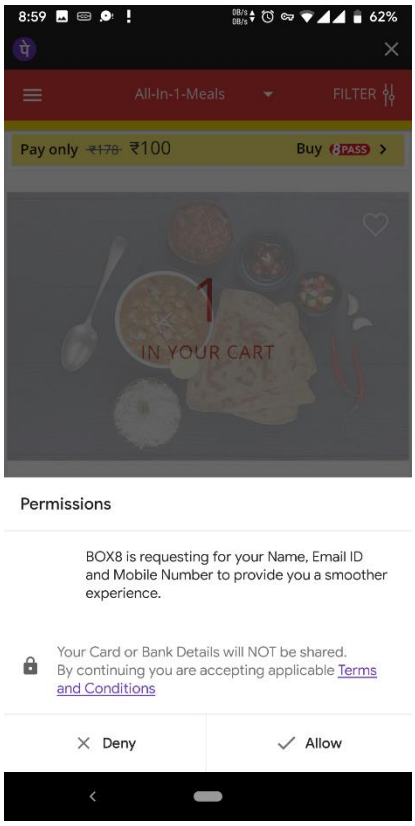

Fig 9.5

Fig 9.6

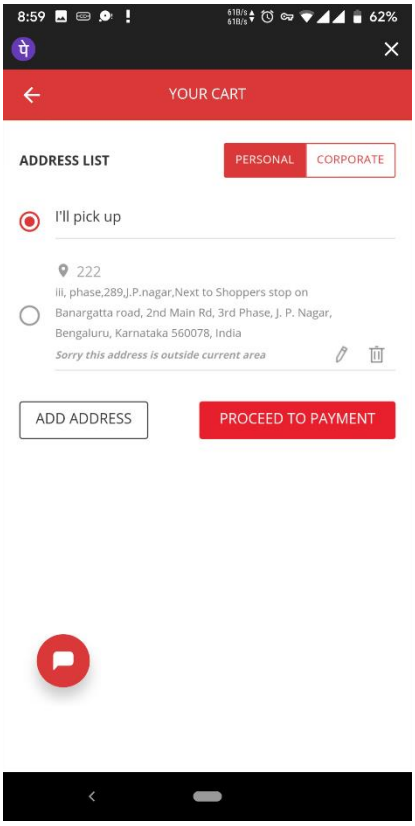

Fig 9.7

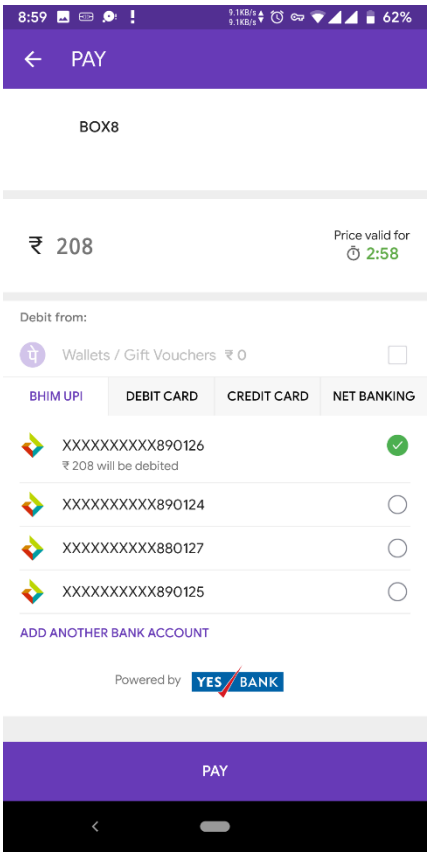

Fig 9.8

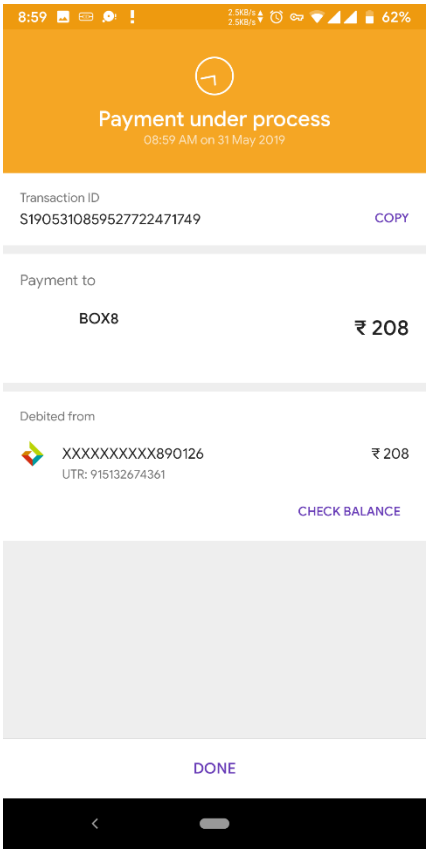

Fig 9.9

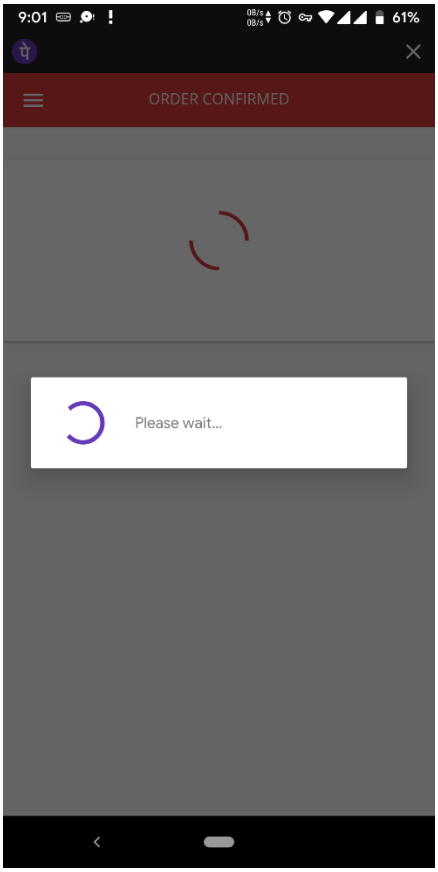

Fig 9.10

CONCLUSION AND FUTURE WORK

All the expectations and requirements by Box8 was fulfilled in time using top notch framework and my coding expertise maintaining the benchmark standards. The internship work has been accepted and is already in the production environment.

There is no much room for future work as I have completed end to end integration. Future work is subject to changes and newer practices that may be introduced by the payment service providers (PhonePe, Paytm, Axis, PayPal, etc)

CHAPTER 11: REFERENCES

11.1. Google Pay APIs for India:
https://developers.google.com/pay/india/api/otherapis/omnichannel

11.2. PhonePe InApp Integration Documentation:
https://developer.phonepe.com/docs/brief-step-by-step-guide

11.3. Gitlab Documentation: https://docs.gitlab.com/ee/README.html

11.4. Redis Documentation: https://redis.io/documentation

11.5. Elastic Search: https://www.elastic.co/guide/index.html

11.6. RSpec Documentation: http://rspec.info/documentation/

11.7. PostgreSQL Documentation:
https://www.postgresql.org/docs/10/index.html

Date: 28th May 2019

To Whom It May Concern

We are glad to inform that **Mr. Janmejay S Purohit** from **Dayananda Sagar University** is undergoing internship at Box8 in Payments Team. His tenure of internship is from **15th January 2019 to 15th July 2019.**

During his tenure as Software Development Intern he has worked on various projects like

- Google Pay Omni Channel Payment Integration
- Refund Status Check from Box8-POS
- Refund Integrity Worker
- PhonePe In-App Integration (On-going)

We found him extremely hard working, dedicated and inquisitive. His association with us has been very fruitful.

For Poncho Hospitality Pvt. Ltd.(Box8)

Abhishek
Tech Lead, Payments Team
+91-8092200694
abhishek@box8.in