B.TECH. PROJECT REPORT

On

Full Stack Development Intern

BY

Rishabh Kumar Verma, 160001048

DISCIPLINE OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY INDORE

November, 2019

Full Stack Development Internship

PROJECT REPORT

Submitted in partial fulfillment of the
requirements for the award of the degrees

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by:

Rishabh Kumar Verma, 160001048,

Discipline of Computer Science and Engineering,

Indian Institute of Technology, Indore

Guided by:

Dr. Bodhisatwa Mazumdar,

Assistant Professor,

Computer Science and Engineering,

IIT Indore

INDIAN INSTITUTE OF TECHNOLOGY INDORE
November, 2019

CANDIDATE'S DECLARATION

I hereby declare that the project entitled **"Full Stack Development Internship"** submitted in partial fulfillment for the award of the degree of Bachelor of Technology in 'Computer Science and Engineering' completed under the supervision of **Dr. Bodhisatwa Mazumdar, Assistant Professor, Computer Science and Engineering**, IIT Indore is an authentic work.

Further, I declare that I have not submitted this work for the award of any other degree elsewhere.

Rishabh Kumar Verma

CERTIFICATE by BTP Guide

It is certified that the above statement made by the student is correct to the best of my knowledge.

Dr. Bodhisatwa Mazumdar,
Assistant Professor,
Discipline of Computer Science and Engineering,
IIT Indore

PREFACE

This report on "Full-Stack Development(Internship at Dunzo)" is prepared under the guidance of Dr. Bodhisatwa Mazumdar, Assistant Professor, Computer Science and Engineering, IIT Indore.

Through this report, I have tried to provide a detailed description of what projects I've got the chance to contribute in during internship. I also tried to design approaches for future improvement.

I have tried my best to explain the proposed solution.

ACKNOWLEDGEMENTS

I want to thank my team **User-Backend** for their guidance and constant support in structuring the project and providing valuable feedback throughout the course of various projects. Their overseeing the project meant there was a lot that I learnt while working on it. I thank them for their valuable time and efforts.

I am grateful to my mentor **Mr. Shubham Agarwal** without whom these projects would have been impossible. He provided valuable guidance to handle the delicacies involved in the project and also taught me how to write a scientific paper.

Lastly, I offer my sincere thanks to everyone who helped me complete this project, whose name I might I have forgotten to mention.

Rishabh Kumar Verma

Contents

List of Abbreviations

API	**A**pplication **P**rogram **I**nterface
DB	**D**ata**B**ase
OTP	**O**ne **T**ime **P**assword
COD	**C**ash **O**n **D**elivery
B2B	**B**usiness **T**o **B**usiness

Chapter 1

Introduction about Dunzo

1.1 Background

Dunzo is a startup in Bangalore, India. Their main goal is that if anything is getting delivered from point A to point B, then it should get delivered by Dunzo. They are currently functioning in cities like Bangalore, Hyderabad, Chennai, Jaipur and 5 more cities. As of now they are delivering in categories like food, grocery, medicine, Pick Drop and Pillion.

Dunzo became a popular startup for their unique idea which was nowhere implemented earlier. Like Flipkart and Amazon delivered household items, Uber and Ola focused on transportation only, Swiggy and Zomato focused on food delivery only. But no one actually focused to do everything on same platform and that should be done because at the core, each of them are delivering something, so it shouldn't be a problem if we're delivering things which doesn't fall in same category. And now-a-days Amazon has now realized this fact, so Amazon has started delivering grocery, and they're in position to buy Uber Eats.

Beside of delivery things from shops for us, Dunzo became popular for one thing on which no was focused, and to bring some competition in this field Swiggy too started doing a bit. Dunzo completes a very useful case of **Pick and Drop**, say, you've forgot your charger at your home, and you came office, so you want it to get deliver from your home, you can just Dunzo it. Just give your pickup and drop locations, and that's it. Dunzo will do that task for you. Beside this, you can order literally anything via Dunzo, if that area is in serviceable range of Dunzo, you can give pickup location of shop and the items you want from the shop, Dunzo partner will pick that up for you.

These were some of the reasons Dunzo got popular and **Google** were very excited with this idea, and that's how they became the India's first start-up in which Google gave the funding.

Talking about how Dunzo functions internally is that it's divided into three main teams, and that are **User, Merchant and Partner.** I was the part of User team which mainly focused on back-end development. User team is further divided into 4 different sub teams and they are: Payments, User Promotion, New User Activation and B2B merchants.

1.2 Internship Experience

In these 6 months of internship, I got to work on various aspects of Dunzo. Mainly it was around Invite Code, Dunzo Cash, offers/promotions and a payment method LazyPay. I got to work with front-end team as well, for the tasks which had front-end dependency as well. At the time I was in payment team, we were actually planning to integrate a new payment method to provide some more ease to users for paying on Dunzo. And there was one more thing that was pending from a lot of time, that if any payment method was going down in terms of successful transaction rate, they manually had to turn off that payment method in order to stop users making payment through that payment method. So, we were planning to have a system which would automate the task pf turning off and on for these payment methods.

In User Promotions, we were actually focused on how to increase users activity on app by making them use more offers and coupons and Dunzo cash. Main problems which we faced was loading offers on Home Screen was taking huge amount of time, which further was a very bad user experience as Home Screen was the first screen user will view and if that will take this much time then it was not at all a good experience, so we had to optimise that time.

Dunzo Cash is nothing but a first-of-its-kind rewards ecosystem designed to provide benefits to Dunzo customers. Dunzo Cash can be used in any transaction on Dunzo at some extent. Dunzo cash was mainly based on Dunzo Cash Policy which was defined at various levels, which was earlier

Chapter 2

LazyPay Integration

The following chapter discusses about what is LazyPay, why was LazyPay in Dunzo, major APIs used in integration, what were results after integrating this and what are future scope in Payments.

2.1 About LazyPay

LazyPay is a **pay-later** payment method, which is very popular these days. It's different from conventional payment like UPI, Paytm, Debit/Credit Card. It's a credit tool which has many advantages over conventional tools.
The advantages of credit tools over conventional tools are:

- They involve only two bank servers comparing to conventional tools, as they involve at least of 4 servers. Involving more servers means more chance of failure because if any of them is facing any technical issue, that maybe because of high traffic, heavy requests, or any other reasons, then there are very high chances that transaction may get failed in between. Let's try to understand the situation through an example, say **A** and **B** are two clients trying to send money online via UPI, which is very popular payment method these days, in which **A** is trying to send money to **B**. So, first A will request to A's payment mode server which is using UPI for transactions. Some of these payment modes are Paytm, GPay, AmazonPay, PhonePe etc. Say in this case, it's GPay. So A will request to GPay's server that I want to send **x rupees** to B. GPay will request A's bank server to give **x rupees** to it. And then A's Gpay will transfer that amount to B's Gpay, further B's Gpay will request B's Bank to accept the payment of **x rupees**. So, we can say that if anywhere request got failed then transaction will get failed.

- In case of money is deducted from one end but somehow it was not able to reach at the other end, due to problem in bank servers. Then the cron which periodically checks for such cases runs usually in 7 days for bank servers in most of the cases. So, there are chances that you finally get your money back in such cases atmost after 7 days. But in LazyPay and similar payment methods, they involve only merchant's server and lazypay's server for the transaction, so first the chances of getting a transaction failed is less than from previous cases, and say if it still does then their cron runs after 2 days. **Cron** are basically functions which are called by server periodically.

- At the same time they reduces the instant involvement of bank server's which further reduces the load on bank servers.

- As they're **pay-later** kind of payment method so they'll ask you to pay for the transactions you've done using them in last 15 days or some fixed period, depending on your credit score. How much you can pay using them is also dependent on your credit score.

2.2 Reason for integration

Before LazyPay there was no payment method in Dunzo, which had credit based pay-later system. And with LazyPay already being a popular payment method. So, it was a good payment method to integrate considering it's very high successful transaction rate(more than 90 percent).

2.3 APIs used in integration

2.3.1 API flow for non registered user

Flow for non registered has done in following way:

- Eligibility API – This API is used to check user's eligibility based on his credit score, and returns whether user is eligible or not to make a transaction via LazyPay. Credit score generally takes into account how often you make transaction through LazyPay, and of what amount mostly.
- Token Initiate API – If the user is eligible then this API is getting called which sends an OTP on user's phone no. provided.
- OTP Validate API – This API is used for validating the OTP entered by user to the OTP they send while calling Token Initiate API
- Initiate Pay API – If everything went fine as of now, then this API is used for initiating the transaction, which further results into creating the task, if transaction got successful.

2.3.2 API Flow for registered user

Flow for registered user is in following way:

- Eligibility API – This API is still used because user may exceed the max. amount he/she can make in transaction. So, it's always used before making any transaction.
- Initiate Pay API – If the registered user is eligible then we call this API to initiate the transaction.

2.3.3 Some other useful API

Other useful APIs which can also get used in transactions are:

- Status Check API – At any point of time, what is the status of any transaction can be obtained using this API.
- Refund API – In case of something bad happens which resulted into LazyPay paid on user's behalf but somehow task was not raised, then we've to refund that balance.
- Unlink LazyPay API – If user is not going to use LazyPay then he/she may unlink LazyPay from Dunzo.

2.4 Results

LazyPay integration proved to be a one of the most used payment method in Dunzo. It's the most preferred payment method of 10 percent of Dunzo's user. Within 4 months after integration almost 2 lakh task were paid by LazyPay, with almost more than 90 percent success rate. Share of post-paid task was nearly 9 percent of total task raised.

2.5 Future Scope

API like Eligibility API are called like twice whenever user reach to payment page, that can be reduced to once, if the response of first call can be cached. That will save lot of time simultaneously making response of home page a little bit faster.

Chapter 3

Circuit Breaker

The following chapter discusses about a problem of manually turning off and on any payment method, if goes down or up. And how did we reach to the solution, it's impact on production.

3.1 Problem

We've many payment method for example Paytm, Amazon Pay, GPay, Simpl, LazyPay, UPI and COD. Leaving COD, each of them is online payment method. So, they suffer many times ups and downs due to third party servers. There are chances like if so many requests are coming, or there is something wrong with their back-end or maybe something else responsible for their server down. It'll cause most of the transaction failed. So, it's our responsibility to stop users making payment from those payment methods. So, for such situations earlier we have engineers/analyst assigned for the task, they used to keep an eye on every payment method about their success rate, and if they are going below the threshold rate, then first they stop making some users to make payment, if still the result is same, then they manually shut that payment method for a while. And they keep checking after sometime, that the issue has been resolved or not. Doing such task manually is a very hard and tedious job to do. Our goal was to automate the system in a way that it automatically identifies the payment method to shut down or when to let them function again.

3.2 Solution

Solution for the given problem can be thought in following way:

- We made a cron whose job is to track all the transaction done in last 15 minutes.
- After having the details of every transaction done in last 15 minutes, we segregate them on the payment mode. Like all transaction made through lazypay at a place, all transaction made through paytm at a single place and respectively. Then we analyse them which payment method has less than 60 percent success rate, then we made them go into **Half-Open** state. If any of them has less than 50 percent success rate, then those payment mode are made to go into **Open** state. And a mail is generated to respected team, that this payment method is going down.
- After every 10 min, for the payment methods in **Half-Open or Open** are made to go into **Closed** state, for 10 transactions, to check if the server is up or not. If the

success rate is above the threshold value, then they are kept into **Closed** state, else depending upon their success rate they're forced to go into **Open or Half-Open** state.

- Doing this way, we made the job of one person done by Dunzo's server itself. And the task was earlier so difficult to manage now became easy, as now no one has to take care of how and when which payment method is going down.

3.3 Further improvement

There is now this provision to improve the system a bit. The algorithm we use in calculating the success rate of any payment method is done by counting no. of successful transaction divided by total no. of transaction by that payment method. This was giving a little bit wrong result because if we're checking transactions of last 15 minutes then there are chances that transactions done in minutes between 14th and 15th are getting failed due to which we're saying that this payment method is suffering server down but transactions done in 1st 5 mins. are good, then this defines that server went down in 14th to 15th minute, but it went up again very soon, as we can see the results in early 5 minutes. So, instead of weighing every transaction same, we thought to give more weightage to transaction which happened lately more to transaction done very early, doing this way it'd be fair to make the decision that whether or not to make the payment method shut down. For this method to implement we were planning to use Weighted Arithmetic Mean for development.

3.4 Weighted Arithmetic Mean

The weighted arithmetic mean is similar to an ordinary arithmetic mean (the most common type of average), except that instead of each of the data points contributing equally to the final average, some data points contribute more than others. The notion of weighted mean plays a role in descriptive statistics and also occurs in a more general form in several other areas of mathematics.

3.5 Result

Using weighted arithmetic mean, result now will be more reliable than earlier, and the system was so smooth that there was no dependency on anyone. So, even if some payment method going down in night, now there is no problem, as now day and night are no different.

Chapter 4

Cohort Optimisation

The following chapter discusses about how new cohorts are created, how they helped to the system, and if not done what can they effect

4.1 Offers

User-Promotion team generally has to take care of offers and promotions. And how offers are created and which users are needed to see those offers are very interesting problem to solve. So, Dunzo give offers to attract new and existing users to raise more and more tasks from Dunzo. By this way we increase their involvement on the app. If we're creating any offer we may want that to be visible to users who may actually get benefit from that. So, it was very necessary for us to classify users among groups, who share some common properties or have some commonalities.

4.2 Cohorts

Cohorts are nothing but a categorisation of users based on some characteristics they share with each other. These characteristics can be anything like users who've ordered only medicine in last 7 days, users who have raised only food task in last 15 days (or in last 7 days). So, they can be categories based on such facts and in last some interval of time. They generally are classified on the basis of type of task they do and the interval of time they take to raise the task. Based on that we create cohorts for them, and use them to show offers which are beneficial to them and for us as well.

One more benefit of cohorts are if our offers are targeting to a part of our users then we can see that offers is actually getting used or not by seeing response of those users on that offers. So, in other way, it actually helped product team in proving their speculation about any offers.

4.3 Problem

So we can see that offers and cohorts are related very closely to each other and they most reasonably had to. Offers are visible to users on their home screen, so whenever any users opens the home-page offers and promotions are loaded for that users. So, now since the offers are condition based, so if any users lie in the cohorts defined by

that offer, then they can see that offer, else not.

These conditions are nothing but just a Database query to make and see the result that whether or not they would fall in the given condition defined by the cohort of that offer. Many a time, a offer is not just having a single cohort, they have multiple cohorts underlying them, so a user is keep getting checked for all cohort unless any of them matches.

This was resulting into huge of time, as every DB query has its own time of execution, and sometimes they involved joins of tables, that makes them a very heavy DB query, resulting into increase of response time of Home Screen. Therefore the problem was to minimize that time taken by the Home API to load the offers, as less as it could.

4.4 Solution

4.4.1 Redis Cache

Introduction to Redis Cache

Redis is an open source (BSD licensed), in-memory data structure store, used as a database, 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, LRU eviction, transactions and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster.

You can run atomic operations 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 your use case, you can persist it either by dumping the dataset to disk every once in a while, or by appending each command to a log. 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.

Why Redis Cache

- It is blazingly fast! After all, it has been written in C.
- It's a NoSql Database. That's Amazingly amazing!

- Currently, it is being used by tech-giants like GitHub, Weibo, Pinterest, Snapchat, StackOverflow, Flickr.
- In order to save your cloud database calls and eventually saving some dollars out there, you can of course opt for caching so the Redis.
- It is Developer friendly and by that I mean to say that Redis is being supported in most of the languages (Perks of using an Open Source Technology). Languages like JavaScript, Java, Go, C, C++, Python, Objective-C, PHP and almost every famous language out there has support for this.
- Last and probably the very obvious point, it is open source and stable, so yeah that's another thing to say 'Yes' to Redis.

4.4.2 How Redis Cache works

Let's assume a system where you constantly need a data for awhile but that data is being obtained from Database, by of course running a SQL query every time.

So, let's say the connection time to the DB is some *x secs.* and running query takes *y secs.*

And say we needed the value for 5 times in a while. Then the total time required will be, assuming we have a persistent connection, *5 * y secs.*.

Now, instead of querying over DB, we started querying on redis cache, connection time will be again here, so for the understanding let's say it has the same connection time i.e *x secs.* but querying in redis cache is way faster as compared to querying over DB, because here we just have to find the key where the data is stored which can be searched in O(1). And since this data in RAM, so the read/write here is very fast as compared to read/write in disk. What I meant to say is here querying takes almost *z secs.* where $z \ll y$. So total time taken will be $x + y + x + 4*z$, because for the first time, there won't be data in cache, so in order to restore the data in cache we need to actually make the DB query. And then we have to add connection time for both redis and DB.

This isn't just about speed! Imagine what difference it would make to your users and how much money you could save on server costs.

4.5 Redis Over Database

Querying over Redis is way faster than querying over database, because of how data is stored in redis, and the way it is stored in redis. For getting any value from Database, we have to make a SQL query for the same, and since the data maybe scattered it may involve join of table as well. But in redis since values are stored in form of key value pair, and because of the data stored in this manner retrieval of data for any key is in O(1). And this value can be anything as redis supports many kind of Data Structure such as strings, lists, hashmap.

So, we actually used redis hashmap for our solution.

Redis key was generated in a manner *userId:44989* and the value in correspondence of such key would be hashmap, and it would comprises of various properties of users like count of total tasks, count of medicinal tasks, count of grocery tasks and likewise. So, instead of multiple querying over database, we can just make a single query on redis which would be very less time taking in comparison to a single query on Database. So, we can have all those properties at a single place with very less amount of time. So, we can get all property's value with a single query, which was earlier done using multiple query and that too on while loading Home Screen. So, decision making reduced from almost 2 secs to 10 ms.

4.6 Result

Now we know that querying over Database is way slower than querying over redis, and that was causing the major time taken by the Home API. So optimising that surprisingly reduces the response time of Home Screen from nearly 4.5 secs to 2 secs. And at the same time there was risk that if this optimisation had not been done then response time of Home API kept on increasing which would may in future results into crashing of app. So, we've successfully created a system where creation of offers and promotion won't be creating any problem if it is going to get sticked to this system.

4.7 Points to be taken care of

Jumping over one existing solution from another may results into some serious issues. Like earlier, every update of any task is directly done to Database, and now if we're going to be reliable on redis cache, then we've to make sure that the status of task updation in redis should be done correctly, because of so many factors.

Factors which may cause such things to happen are:

- Retrying APIs may results into update cache so many time, so it should be taken care of by using set in redis, as it store only unique values.
- For the first time, since there will be no data in cache, so we've to run a script which will back-fill the cache with user's data. So, that there shouldn't come a huge request on DB.
- And whenever there is cache miss for any user, we had to make sure to return default value at that time because calling DB for the value at that time may results into large time of execution, so we called an async. function which will back-fill the cache for the next time.

Chapter 5

New B2B Merchant Fraud

The following chapter discusses about the fraud New B2B merchant used to do on Dunzo, how did we find solution for that, and what result came after through.

5.1 About B2B Merchant

B2B merchants are nothing but the traders in the market who have something to deliver from one place to another but they really don't want to bother about how delivery is taking place but that should be reliable and trustable. All they want is to use Dunzo's service for delivering their goods from one place to another.

5.2 Scheme for New B2B merchant

Dunzo always welcome new users by giving them good offers or help in trading in some other ways. Here as well, Dunzo gives 3 free tasks to new merchants to complete. And Dunzo offers merchants to either deliver their goods now or they can schedule tasks as well, so a partner will be available at the time they've scheduled for the task.

5.3 Flaw in scheme

Dunzo offered 3 free tasks for new merchants. But the check for the next task to give them free or not is done by checking whether user has **completed** 3 tasks or not. If not then next task will also be give free to them. In this way they can schedule as many as task as they can and that too forever. After completion of 3 tasks only they will be charged.

5.4 Solution

So the solution thought for the above fraud rather than checking on 3 completed task the check should be on 3 Active + Completed task.

So, in redis cache for merchants users only we maintained a count of active tasks for that user and whenever any merchant task status is either created or cancelled or any potential status change is happening, it's status is also updated in redis. So, now we've active task count of any user in redis and we can check if the count of active + completed

task is less than 3 then we can free the next task else we will be charging for the next task.

5.5 Result

This New B2B merchant fraud came down to 0 percent as it was happening earlier, which nearly saved 2 lakhs rupees in the coming month as of expected.

Chapter 6

Dunzo Cash usage at various levels

The following chapter discusses about what is Dunzo Cash, what is Dunzo Cash Policy and how it was used earlier, and it is going to be used now and what future demands were also coming.

6.1 About Dunzo Cash

Dunzo Cash is a first-of-its-kind rewards ecosystem designed to provide benefits to Dunzo customers. These benefits can be taken while you're using Dunzo's service in which already there has been no offer used. Whenever Dunzo cash is applicable it'll pay some part of the total amount you have to pay.

6.2 Dunzo Cash Policy

Dunzo Cash is controlled by various factors and all those parameters are part of Dunzo Cash Policy. Some of the basic parameters already available in Dunzo Cash Policy are like max. dunzo cash per task any user can use, max dunzo cash discount percent per task. Users can get dunzo cash while they've signed for the first time or they can refer the app to some other user, by this way they can also earn Dunzo cash. So, how many users a particular user can refer and earn dunzo cash depends on the policy currently applicable for the user. And when dunzo cash will get expire for a user is also controlled by the policy itself.

6.3 Dunzo Cash Policy level

So the policy which will finally be applicable on user have some levels if higher level of policy is active for the user then he will have advantages from dunzo cash based on the parameter given in that policy.

Hierarchy of Dunzo Cash Policy are as follows:

- City Level – Policy for the whole city
- Geo Level – Policy for a particular geo in a given city
- Cohort Level – Policy for users who fall in a particular cohort
- User Level – Policy for group of users.

- Default – In case any of them is not applicable.

Earlier User level of policy was not there but the product team found a very useful use-case in which policy has to be at group of users. Like suppose there is a TED session going on somewhere and Dunzo wants to sponsor it and at the same time they want to do marketing about their product as well. So, this was the best place where they can offer Dunzo cash to users and have them controlled through a policy because they can't fall either in a cohort or in a same city or in a same geo. So there has to be a policy at this level.

6.4 Problem

There is a user-policy table in database, columns in this table were policy, policyId, type, typeId and extra-Data. Earlier only three types of policy were there so this table was not populated. But if the user policy will be added to this table then it'd be get highly populated by them, because policies on higher level have larger activation period compared to user level policy.

6.4.1 Why table would get polluted?

Entry of user level policy would be like policy will be user policy, policyId would be some integer, type will be "user" and typeId would be userId. So, let's say if the policy has been made for group of 200 people, then there would be 200 entries in that table. So, we needed a solution for the above scenario.

6.5 Solution

To tackle this problem, we actually took help of redis and Database in combination. We actually created another table with listId and list. ListId would store some integer and list would be group of all those users who belong to a common group session. There is a possibility that at a time a user can belong to multiple user policy which was not the case earlier, because no user can belong to different city or no user can have multiple geos. So, to sort this out we have a redis cache for the user, which will be a set of tuple (combination of different objects) and this will be sorted. Tuple will have elements like activated-on, deactivated-on and policyId. And policyId will be mapped to policy with some other keys. And this set will be sorted based on activated-on field, because the element which have the latest activated-on will be the one user should get benefited from. Since there is two types of cache we're dealing currently with, then we have to tackle for both of them in case of cache miss.

In case of cache miss for the policy and policy-id we can fetch it back from DB. But for the case of user set it would be hard to restore the set of tuple, because for that we've to traverse in each of the policy in DB just to look out is this the policy user was belonging to. If not then skip for the next policy and so on. So, we actually needed to have a timeout for such key quite large, larger than active period of largest active policy. Still there would be possibility that cache miss may happen in that case, we'd

not be concerning about any user policy for that user, if he is belonging to some other policy then he would have advantage of that policy else he would be enjoying default policy.

6.6 Result

With these feature live in production, it was possible for Dunzo to carry on such session with no problem at all, and they can do experiments as well for small small groups.

Chapter 7

Invite Code fraud

The following chapter discusses about Invite Code and fraud users were making through them and how did we find the solution for the above with future help as well.

7.1 About Invite Code

We already discussed about Dunzo Cash and Dunzo Cash Policy as well, so we discussed there that anyone can earn dunzo cash by referring Dunzo someone else as well, and how much he and the one who got referred can earn depends upon the policy. But there maybe the case, that a user started using Dunzo on his own, without being referred by someone, but he also should get awarded for that. So, for that we've Invite Code that code can be entered by user in place of referral code. And he will get some dunzo cash as it had been described in invite code rule.

7.2 Fraud with this

User when knew this feature that they can earn dunzo cash on their own, then they started creating multiple account on the same device and started entering the same invite code and kept earning dunzo cash, as earlier. So this fraud was reaching at next level, as it spread very fast manner, and company was suffering a huge loss for the same.

7.3 Problem

Earlier there was this system that you can create invite code either for All the cities or for any particular city, but not for a bunch of cities, you want neither once created you can edit the city. So, this was the first problem they faced, because the fraud was mainly happening in Delhi, Noida and Gurgaon. So, they wanted to remove these cities from the usage of invite code. And secondly, this was anyway a great fraud, we shouldn't let users create multiple account from the same device.

7.4 Solution

7.4.1 Device ID

A Device ID is a string of numbers and letters that identifies every individual smartphone or tablet in the world. It is stored on the mobile device and can be retrieved by any app that is downloaded and installed. It is most certainly unique for every kind of device whether it is Android or iOS device. So, we started taking into account the detail for every user that what's his device ID, and if the device ID is same, we were not allowing users to create accounts anymore. This reduced the increase in fake users which were earlier very easy and reduced the fraud to some extent.

7.4.2 Front-end changes for multiple cities

Front-end changes were necessary because even from back-end we can make sure that we will now allow for invite code to be created for multiple cities, but the one who is creating invite code will not be running python script and create invite code directly to the Database. So, I did the front-end changes to multiple options rather than choosing only ALL, or any particular city, and earlier only one city-id was sent that would be either any particular city's id or it was NULL for ALL cities, but now front-end will send a list of city-id to the back-end, and from back-end I ran a loop to create invite code for all the cities in the list of city-ids.

7.5 Points to be remembered

As we created multiple cities for one invite code and earlier there was only one city-id for each invite code so, and since we change the system, so we actually had to make sure about backward compatibility. Because earlier wherever they were trying to get invite code from Database using unique city-id that constraint had to be removed and rather than querying for unique feature we have to query if such invite code is available from any of the given city-ids.

7.6 Result

After the changes merged to production, the invite code fraud was brought down from 45 percent to 15 percent. Changes done in production were cities like Delhi, Noida and Gurgaon were exempted from the use of invite code for a while. And a different invite code was created for those 3 cities only. So, this feature was highly in use their as of now.