B.TECH. PROJECT REPORT

On

Full Stack Development Intern

BY

Pushpendra Kumar, 160001046

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:

Pushpendra Kumar, 160001046
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

CANDIDATES' DECLARATION

We hereby declare that the project entitled **"Full Stack Development"** 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, we declare that we have not submitted this work for the award of any other degree elsewhere.

Pushpendra Kumar

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 in Dunzo, Bangalore" is prepared under the guidance of Dr. Bodhisatwa Mazumdar, Assistant Professor, Computer Science and Engineering, IIT Indore.

Through this report, we have tried to provide a detailed description of our approach, design, and implementation of an innovative projects I got the chance to work in. Some of them were optimisation over the system and some of them were new feature which I had to implement. I have tried to discuss them in depth as much as it was possible.

We have tried our best to explain the proposed solution, along with the detailed analysis of our features and fraud which was there in system and how we tried to remove them from the system.

ACKNOWLEDGEMENTS

We want to thank our B.Tech Project supervisor **Dr. Bodhisatwa Mazumdar** for their guidance and constant support in structuring the project and providing valuable feedback throughout the course of this project. His overseeing the project meant there was a lot that we learnt while working on it. We thank him for his time and efforts.

We are grateful to **Mr. Akshay Megaraj**, without whom this project would have been impossible. He provided valuable guidance to handle the delicacies involved in the project and also taught us how to write a scientific paper.

We are grateful to the Institute for the opportunity to be exposed to systemic research, especially Dr. Bodhisatwa Mazumdar,.

Lastly, we offer our sincere thanks to everyone who helped us complete this project, whose name we might have forgotten to mention.

Pushpendra Kumar,
B.Tech. 4^{th} **Year**
Discipline of Computer Science and Engineering,
IIT Indore

Contents

List of Abbreviations

DB	**D**ata**B**ase
API	**A**pplication **P**rogram **I**nterface
mins	**M**inutes
UI	**U**ser **I**nterface

Chapter 1

Home Screen Reordering

The following chapter discusses about problems we were facing on Home Screen regarding widgets, what is widgets and how did we actually solve the problem we were facing and the result after that.

1.1 About Widgets

Widgets are nothing but just a banner which appears on app, they are used to display offers, promotions and some times they're used to show about new products we many are going to launch in future. So, basically they're nothing but just a banner which appears on app, and has these many uses.

1.2 Home Screen

Home Screen, is actually nothing but just a collection of widgets in some order, these widgets generally differ with each other in terms of length and breadth and sometimes they vary with what they are trying to show, like some are showing popular stores in your city, some are showing offers and promotions, some are showing status of partner availability.

1.3 Problem

The problem was the ordering of these widgets on the home screen was hard-coded in the back-end, so in order to have any changes in the way these widgets appear on home screen we actually needed to change the order from the back-end and make a deployment. To these changes to finally being displayed in the production app, we needed to make a deployment as well. The whole process was very cumbersome, inefficient and huge time taking.

The need was to have a system which would do this process, like some configuration change in database or something like that, but that shouldn't ideally involve any kind of hard-coded part and most importantly any deployment. As updating in database would be very fast and can be seen in app very fast.

1.4 Solution

We finally decided to give ids to each widget, so now every widget can be identified by its unique id. And there will be a table in DB, which will which have idea about what contents are going to be shown in any particular widget. And there will be a separate table which have columns like city-id, area-id, order of widgets and screen type. So, now for every city we can have different pattern of widgets, that can vary even on area level as well.

Now, we actually needed a back-end support for the above proposed solution. So, we needed to create significant APIs for the same.

Following are the APIs we made to support the above solution.

- Create New Widget Order for city and area
- Update any existing Widget
- Get Home Screen Order for any city/area
- Delete any existing configuration

1.5 Major APIs

1.5.1 Create New Config.

This API will be used for making a new order of widget on the home screen. It will be expecting parameters like area-id, city id and order of widget in the request. This API will first authenticate the request before saving in the database. Some authentication will be like if any id is repeating more that one time in the request, or the number of widgets is more than that required minimum number. After all that we will save the widget order in the database. This will add a new config., which is unique in the table. If something unexpected comes, then will throw an error and won't be saving anything in the DB.

1.5.2 Update Widget

We may need to change any existing widget maybe in future, we may want to create a widget to have some new content getting stored in it. So, there has to be some option to edit the existing one rather than creating a new API all together. The parameter will almost be similar like Create New Config, but instead of creating a new entry it'll update the existing config.

1.5.3 Get Home Screen order

This API will be expecting some city-id/area-id and in response it'll return the ordering of widgets in that city-id/area-id. Response will look like a list of widget-ids in a manner who they will appear on the app.

1.5.4 Delete Config

We may want to delete any config. as per our need may change, so, there has to be an API which will do the following method. It'll actually expect widget-id and will changed the status for that widget-id in DB from ACTIVE to DELETED or anything else.

1.6 UI Support

Since this was a whole new feature, so we actually needed a front-end support for the same. I actually created a complete UI support for the changes done in back-end. I worked on ReactJS in order to do the same .

1.7 Result

After the changes were live in production, it was now very ease for product teams, to make any change on home screen, without making any changes in code or making any deployment. This actually saved a large amount of money and other resources of company. Feature is still in use and seemed a very handy tool to the product team.

Chapter 2

Decreasing Redis Calls

The following chapter discusses about what is redis, why is redis and why excessive calls from redis was taking so much time. How we reached to the solution and what were its impact on the system.

2.1 About Redis

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.

2.2 Why Redis

Querying over redis is way faster than querying over DB. Because of the way data is stored in here. There are some disadvantages as well of redis and that too because of the way data is stored here. As the data here is in form of key-value pair. So, no key value pair is related to any other key-value pair. Unless they're stored in a way that they've some relations with the key. Like we can join tables in DB, to retrieve some more information. But here it's not the case, because there is no such joining concept.

Redis is very fast because finding values against a key is done in O(1) in comparison of finding values in O(n) in tabular form(if proper indexing hasn't been done). And that too has it's own cost. Read and write is also very fast in redis compared to DB because it's in RAM and r/w for DB is done in disk.

Plus values in redis have their own time-limit, which will get deleted, so they can't be supposed as a primary solution. They should always be a secondary solution, supporting the main database. As the key would get deleted we will always have enough space for the new keys to get stored here.

2.3 Problem

Home Screen comprises of many things like Icon-grid, offers, promotions etc. So, earlier all these values were fetched from DB. But then values in DB were so populated that these query started taking so much time, which affected the response time of home API. So, we shifted a bit from DB to redis, initially it helped a lot, situation still is lot better than if it was not for redis. But guys started using it rigorously. After all, it is also a DB, it also take time in responding you for a key. Now, there was overuse of redis.

2.4 Analysis

2.4.1 New Relic

New Relic is a modern tool which has various advantages. It's mainly used to see what is the average response time of any API, what is the worst time taken by any API to respond, what DB query is taking how much time, and how many redis calls are being taken in any API, and likewise. Basically, it provides a lot more feature to completely analyse how your API is working what load is coming to your system. What is the success rate of any API in last few intervals of time.

This tool actually helped seeing us that on Home Screen API, almost ***150 redis calls*** were there, most of them were GET query, which is used for getting a string which is stored against a key.

After analysis we finally get that there were excessive redis calls, more than we required. It almost took 3-4 ms for redis server to respond for a query. It means that around 0.5 sec of time was taken by making redis calls on home screen. So, there was this scope of reducing them and making home screen API a little bit faster.

This 0.5 sec may seem very small but it actually makes the system much faster than expected, because in micro-world everything is happening in micro-seconds. And, this is thousand times larger than a microsecond. It also reduces the load on redis server making room for other redis call to respond quickly.

2.4.2 Post Analysis

After reviewing of code, we find places where redis calls could be reduced, these places were:

- Calling for store data, every time we found a store.
- The way condition is checked in offers and promotions. This was one of the major areas were redis calls were made redundantly.
- Making same redis call across the function calls.

2.5 Solution

Appropriate measures taken after analysis were:

- ### 2.5.1 Store Data

  Dunzo delivery food and grocery, so these category had to deal with stores timing and availability of items in store. So, google has APIs for that, but calling APIs every time we need store details is very expensive. So, we can't do that. Because the data which is stored in cache can be reliable for a least of 10 mins. So, we actually store data in cache with a time limit of 10 mins. And there will be a cron which will auto update the data of store in redis cache after every 10 mins.

  But how the store data is fetched is that we have a store dzid in redis, and we make a query for store-dzid and then we have it's details. But say we've a 3000 stores, then we may have to make at least of 3000 queries in order to have their details.

  So, we actually created a hashmap for all the stores, since that data will be persistent in cache as well for at least 10 mins, we can rely on that data. And the redis key will be *store-data* and value against it will be a hashmap in which key would be store-dzid and value will be store data for the store. We will make a bulk query and will fetch all the details of store in a single query rather than making separate query for each store.

  This implementation significantly reduces the redis calls.

- ### 2.5.2 Cohort condition

  Earlier for every offer and promotion we've cohort condition for them and to fetch that condition we had to make DB query earlier. So, we actually stored that condition in redis. Just to fetch those conditions in a fast manner.

  But here as well, we were making single query for each offers and promotions just to have their cohort condition, rather than making a bulk query and fetching all the cohort condition for each offer and promotion in a single GET query. And since this queries were on Home screen so they actually saved a lot of time om home screen reducing almost 100 redis calls.

- ### 2.5.3 Redis call across function

  Say, we need a values stored in redis in a function, and then we need to call another function which also demands the same value from redis, so we were actually making 2 redis calls for both of them, we just can simply pass that value as some parameters, if the value of parameter is NULL, then we can make a query just to ensure it's actual value, because that function maybe getting called from somewhere else as well, and there they might not need that value.

  Doing this at most of the places nearly reduced 150 redis calls.

2.6 Result

Reducing redis calls by making bulk queries and removing calls which were redundant almost reduces total redis call in a day which was earlier around 10k to nearly to half which was around 4.5k, it actually decreased so much load on redis. Making redis server a bit free for new calls in future to make. It also reduces response time of Home Screen by 0.5 secs, by reducing nearly 100 redis calls in Home API.

2.7 Future Scope

Our goal was to actually make Home Screen as fast as we can do. The zone I found out was excessive redis calls, there were other factors as well, that optimising redis calls, optimising DB queries, making function statement more efficient than earlier.

Chapter 3

Dunzo Cash Fraud

The following chapter discusses about the fraud on Dunzo Cash and how did we try to remove it.

3.1 Dunzo Cash

Dunzo Cash is nothing but a kind of reward system Dunzo gives to their user when they refer the app to someone or they have just started using the app for the first time entering some default code. Using dunzo cash we get discounts on products up to some extent.

3.2 Fraud

We actually have a dunzo cash policy for every user which is on different level, whichever the policy will be active for the user, it'll control the amount of dunzo cash any user will get on referring someone. There was this parameter in dunzo cash policy, but it was not checked while any user referred to some user. So, this was the reason the fraud was happening. Anyone can refer to infinite guys, earning dunzo cash infinitely.

Earlier there was no check on how many accounts any user can have from a single device, guys started creating multiple accounts on the same device and they referred themselves.

This was the major issue, the fraud was increasing at an alarming rate day by day.

3.3 Solution

I actually had to add a method which will calculate the count of user this user has already referred to, and if this count will be more than the count described in policy then referral won't work.

3.4 Result

As a result, fraud happening due to this miss reduces significantly, saving company from a big loss.