Project Report

A Smart Order Management System with automatic employee task assignment and Order Status tracking for efficient interdepartmental synergy.

Technologies:

* Object Oriented Programming with Dart
* Cross platform application development using Flutter
* Firebase Authentication
* Firestore (NoSQL Database)

Srujan S. Mhase

Project time: 7 Days – 16-01-2021 to 22-01-2021

# Abstract

In any enterprise, the efficiency of order management and fulfilment play a major role while directly influencing its key performance indicators. Key processes in the industry are managed in a “supply chain” i.e., several successive steps in the process are directly dependant on personnel responsible for the preceding step. Any mishap, tribulation or a delay may have a rippling effect on all the successive steps and may trigger a slew of Mutually Exacerbating Catastrophes that might be difficult to contain.

Taking cognizance to this issue, I have attempted to build a resilient and effective yet simple solution to dynamically allocate tasks across multiple teams/departments that will result in minimal friction and to create a simple streamlined user interface to view the status of the same.

# Background Information

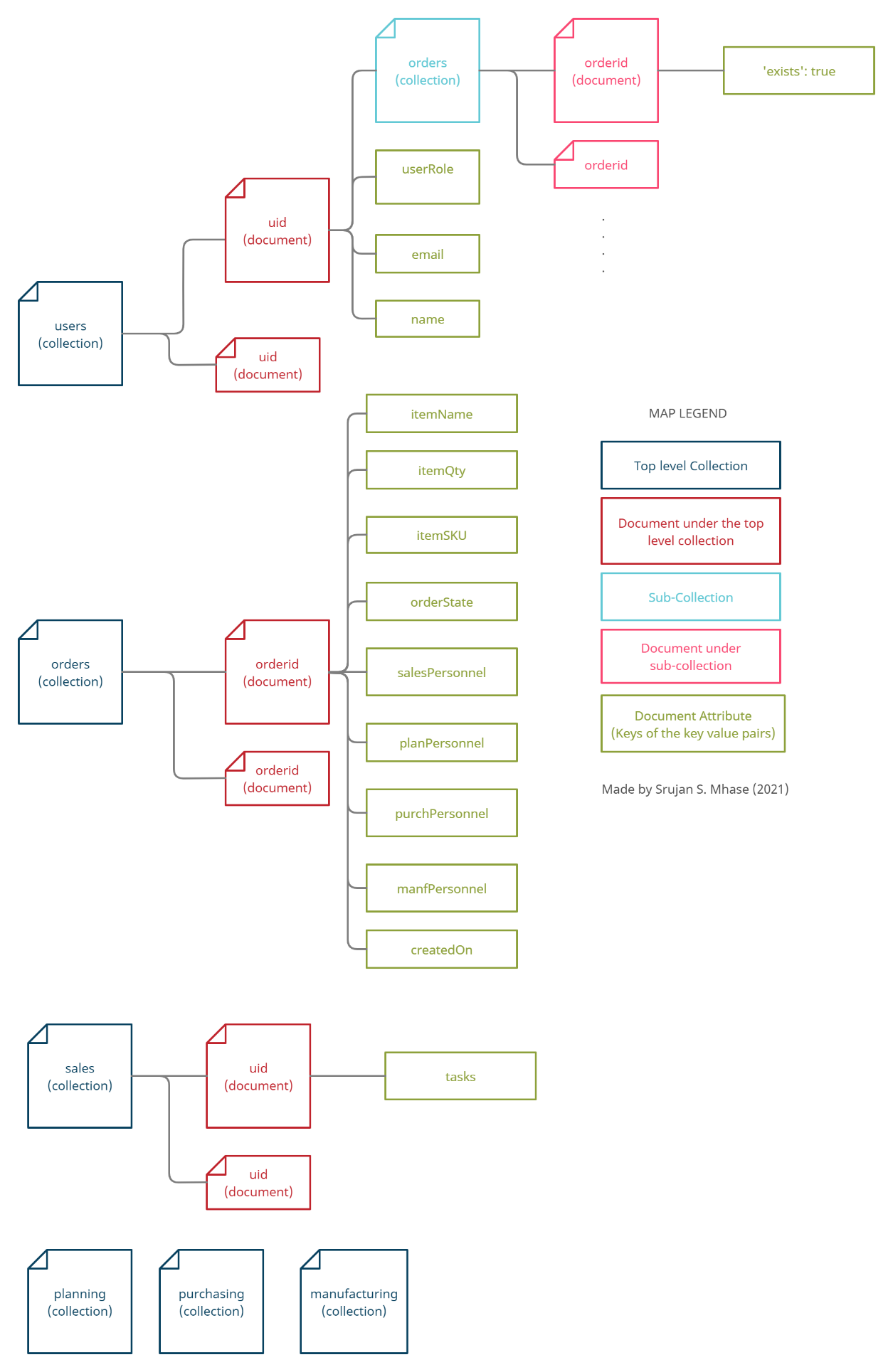
Organisations traditionally operate in “silos” wherein different departments work in their own isolated purviews and inter-departmental synergy, though very much a necessity, remains difficult to implement. The main objective is to enable a single window solution to create new orders, dynamically distribute those tasks to employees across different departments and a streamlined view of their status.

Features

* Create new order
* Dynamically distribute tasks related to the order to employees across different departments with the least number of active tasks at any given time
* Enable a certain level of department specific functionality
* Track the status of the fulfilment process in a single window and easily address any potential problems.
* Manage and edit orders

# Architecture

## Data Storage Schema for NoSQL Database

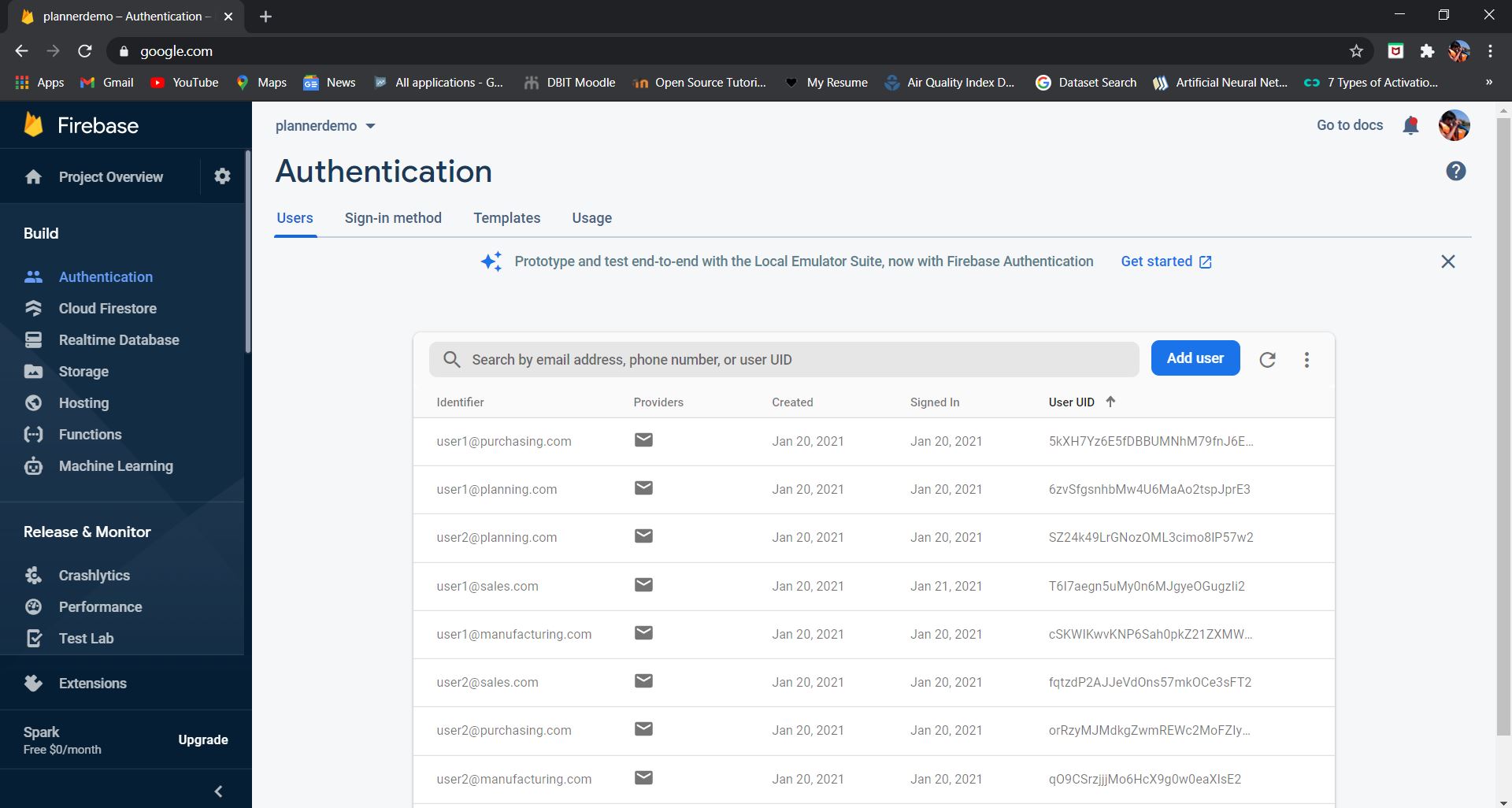


**Fig(1). Data Model - Schema for the NoSQL Data Storage**

NoSQL databases store data in a key value pair format in documents. These documents can either contain key value pair information or may contain child collections or “subcollections.” These subcollections in-turn can contain further child documents; so on and so forth.

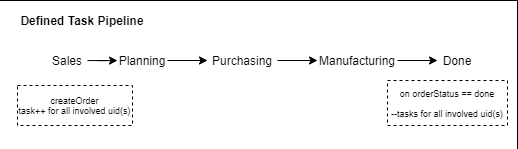
In traditional Relational databases, a convention followed is that data must not be duplicated in multiple locations, however, in NoSQL databases the data might be duplicated in certain situations in order to make querying faster and incur less number of read operations. The figure Fig(1) above is a schema of the database used to store and manage information in this project.

*uid – Stands for Unique Identification of the user. This value can be programatically generated at any situation that may need it for record keeping purposes. In this case, the uid here is generated by Firebase Authentication for authenticating a user.*



**Fig(2). Firebase Dashboard showing the User UIDs generated for each user registered on the platform**

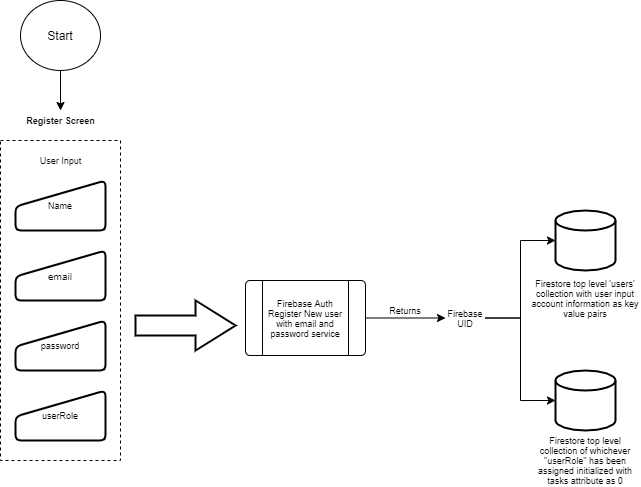
For the purpose of demonstrating, a total of 8 users have been created using the app itself. Two users of 4 departments each. Note the orderState (the current state of the order) is same as the name of the department which is supposed to complete it.



**Fig (3). Shows the assumed task pipeline**

By way of illustration, if the order is being “planned” i.e., the order status is “planning” the “planning” department (the user belonging to the userRole == ‘planning’) must work on the project and mark “planning” as done. When the user does that, the modifyOrder method will update the orderStatus as the succesive one as defined by the task pipeline (shown in fig(3).)

## Registering a new user with a defined user role



**Fig (4) Registering a new user workflow**

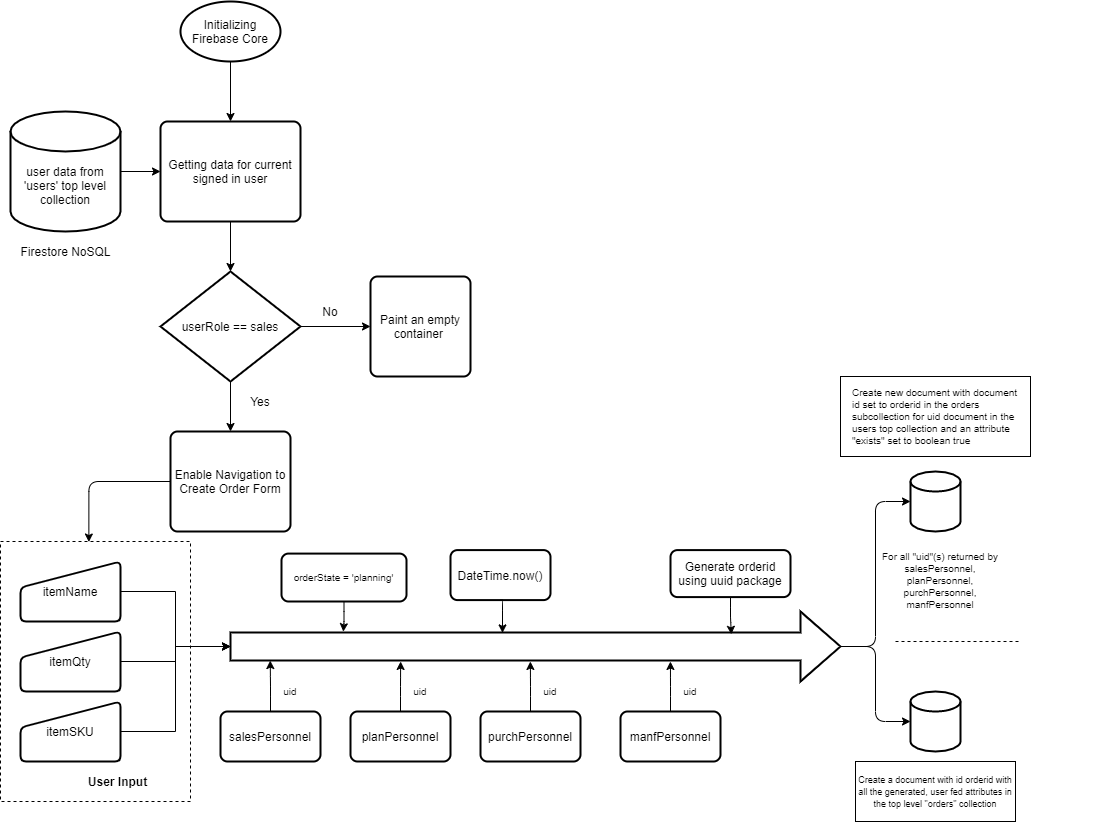
The workflow of registering a new user is illustrated above. As the user input is taken, it is accessed via the textController for the respective formfield and the text is extracted and passed into the firebase authentication’s registration service. The firebase Auth serivice registers the user with the supplied credentials and returns a “uid” unique to the user.

We then use this uid as the document id for storing information in the firestore NoSQL database. An entry is made in the top level ‘users’ collection with the input parameters name, email and the assigned user role; and a secondary entry in the top level collection of whichever user role that the user is initialized as, with the parameter of tasks which indicates the active number of tasks for the user. This value is initialized with int 0.

## Creating a New Order – Process flow

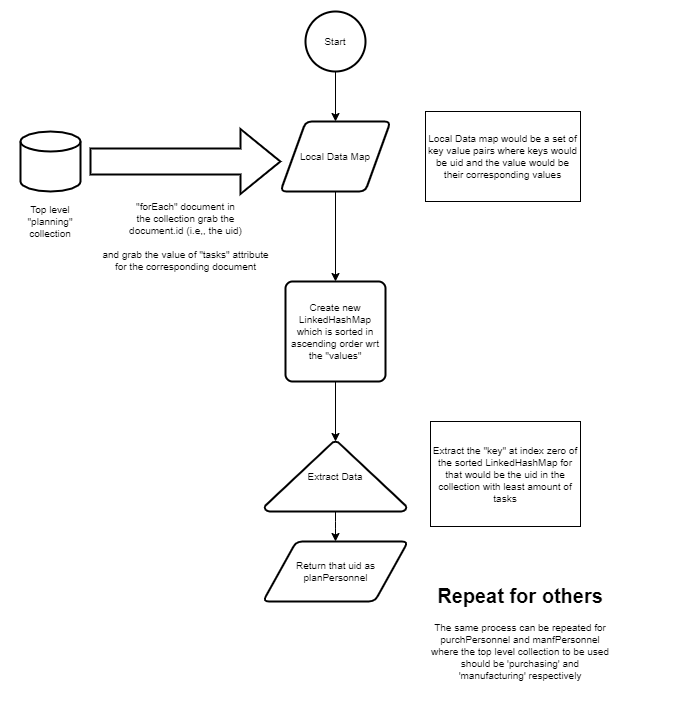
To create a new order, the current user must be assigned with the user role of “sales”. If logged in with any account assigned a role other than “sales”, an empty container will be painted in place of a button navigating to the create order form.

Once the itemName, itemQty and the itemSKU have been entered, the createOrder service will assign the “planning”, “purchasing” and “manufacturing” tasks for that particular order to users with the respective roles. For that, it will access the top level collection of the role for which it is assigning task to for eg. “planning” top level collection, then it will create a local map of all the users registered as “planning” roles in key value pair stores where the key will be the uid(s) and the value will be the value of “tasks” (which indicates the active tasks). Then it will sort the map with respect to the values in ascending order and then return the key of the first index. It will be the uid with the least number of active tasks for that role at that particular time. This process will be repeated for all roles. This is illustrated in Fig(5) and Fig(6).

****

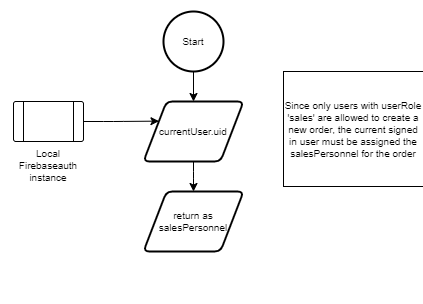
**Fig (5). Shows the workflow process for creating a new order in the system**

The process is illustrated above wherein the sales, planning, purchasing and manufacturing personnel are assigned on the basis of their active tasks. When the order is created, the orderid generated using the “uuid” package for dart, is used as the document id for storing the information of the order in the top level “orders” collection (refer schema). Further, for all the assigned uid (user id’s generated by the firebase auth) for various roles, the ordered document is added with the attribute {“exists”:true} in the the “orders” collection of the uid document in the top level “users” collection. This location is queried for all the orders assigned to the user while displaying it on the main page.



**Fig (6) Generating a list of people assigned tasks across different departments from existing registered users**

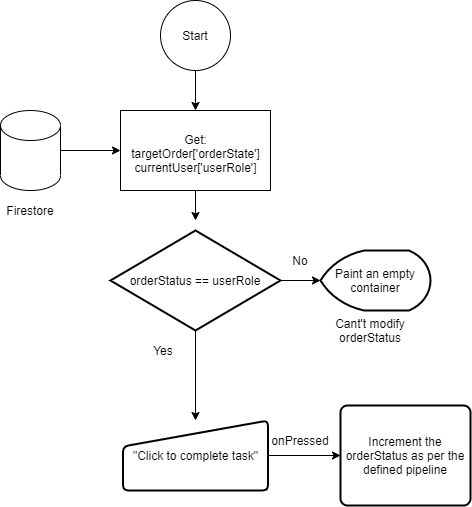
Above flowchart illustrates the process for assigning the uid(s) for each role: planning, purchasing and manufacturing. It details the process workflow of getting all the tasks, mapping the active tasks to the user ids and choosing the one with the lowest active tasks.



**Fig (7) Setting the sales personnel for the task being created**

Since only users with the role “sales” can create new orders, the salesPersonnel for the order being created will be the currentUser.uid

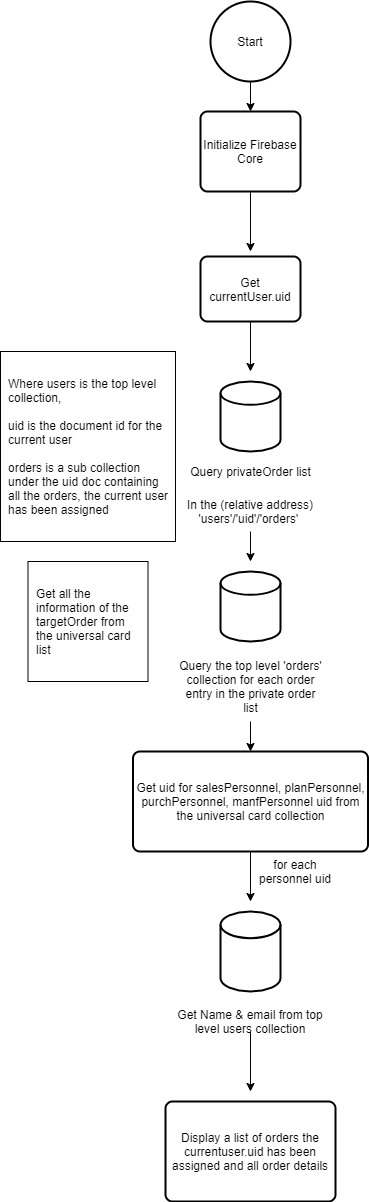
## Updating Order Status



**Fig(8) Process workflow of the order status being updated**

The above figure details the conditions required for the user and the existing orderStatus to be able to modify or update the targetOrder ‘s status. When the conditions are met, the order status is incremented to the successive order status as defined in the *Fig(3) task pipeline.*

## Process to display the order details



**Fig(9) Process workflow to display the orders to relevant users in their main page**

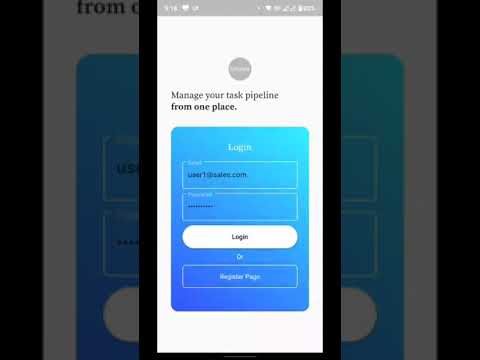
# Results

## Github Repository

<https://github.com/srujanmhase/plannerdemo>

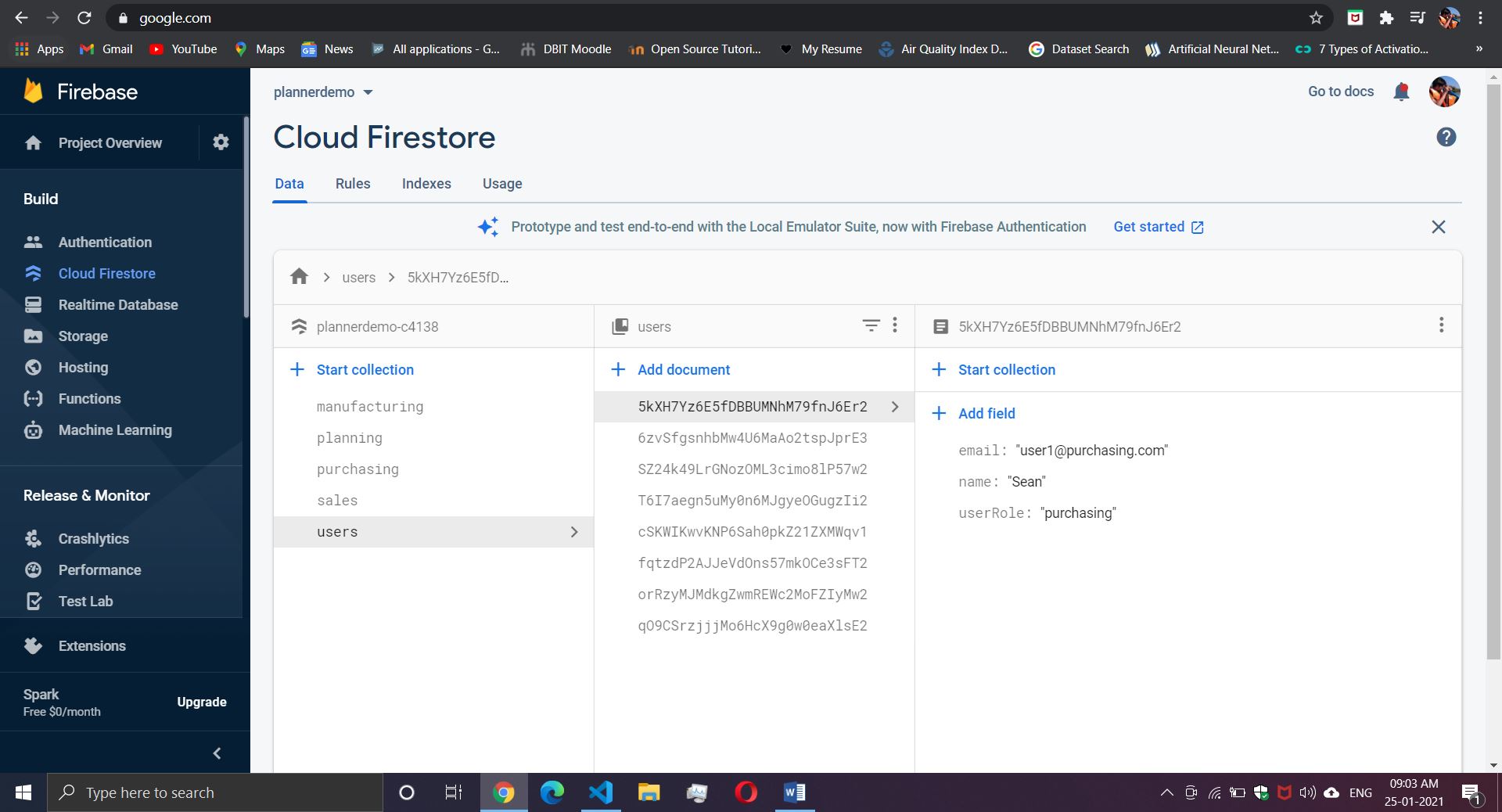
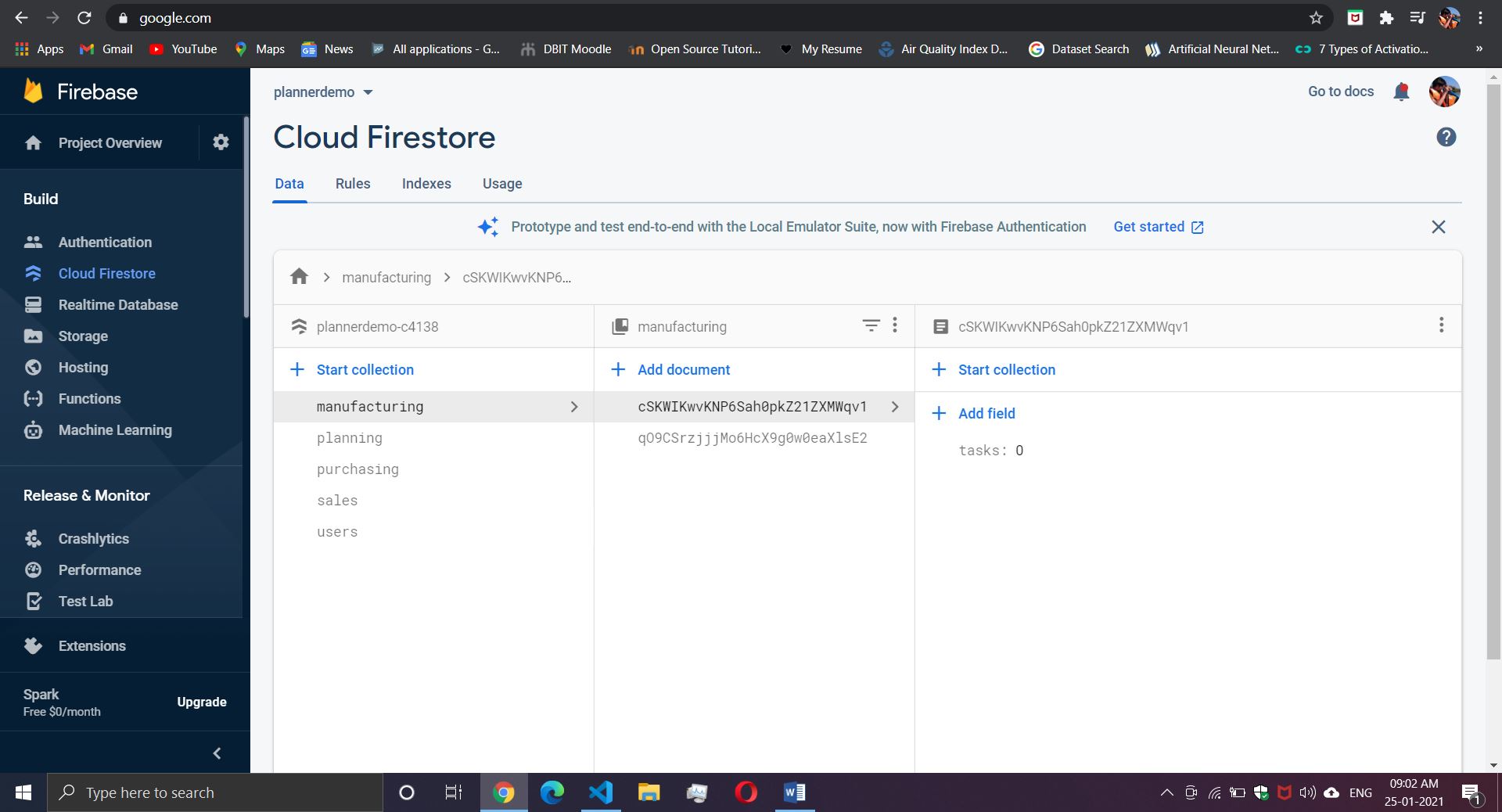
## Video working of the app

*Kindly note that the notification tab screen is merely a user interface concept and hasn’t been linked to active database neither updates dynamically.*

[](https://www.youtube.com/watch?v=plYQncbg21Q)

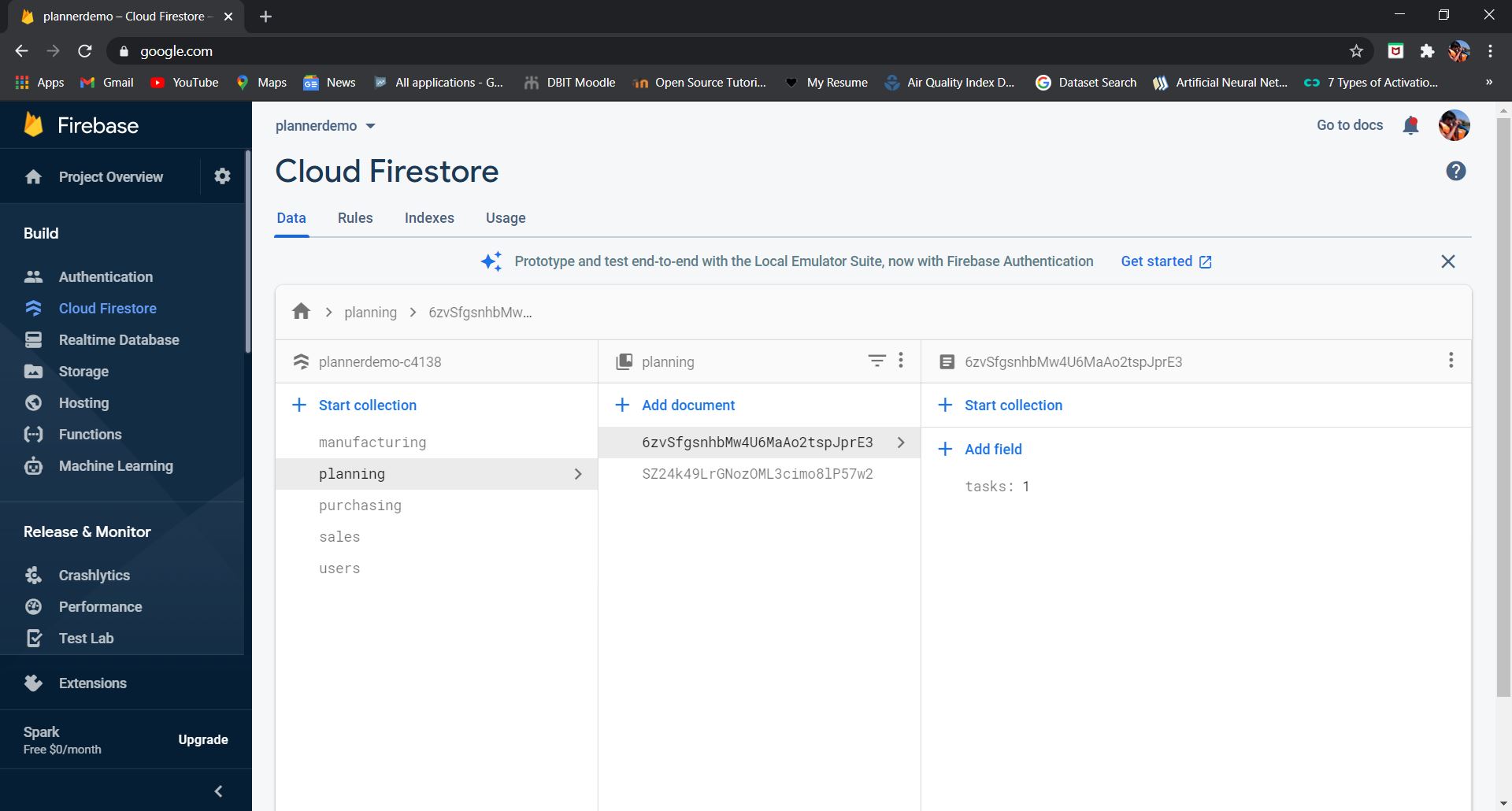
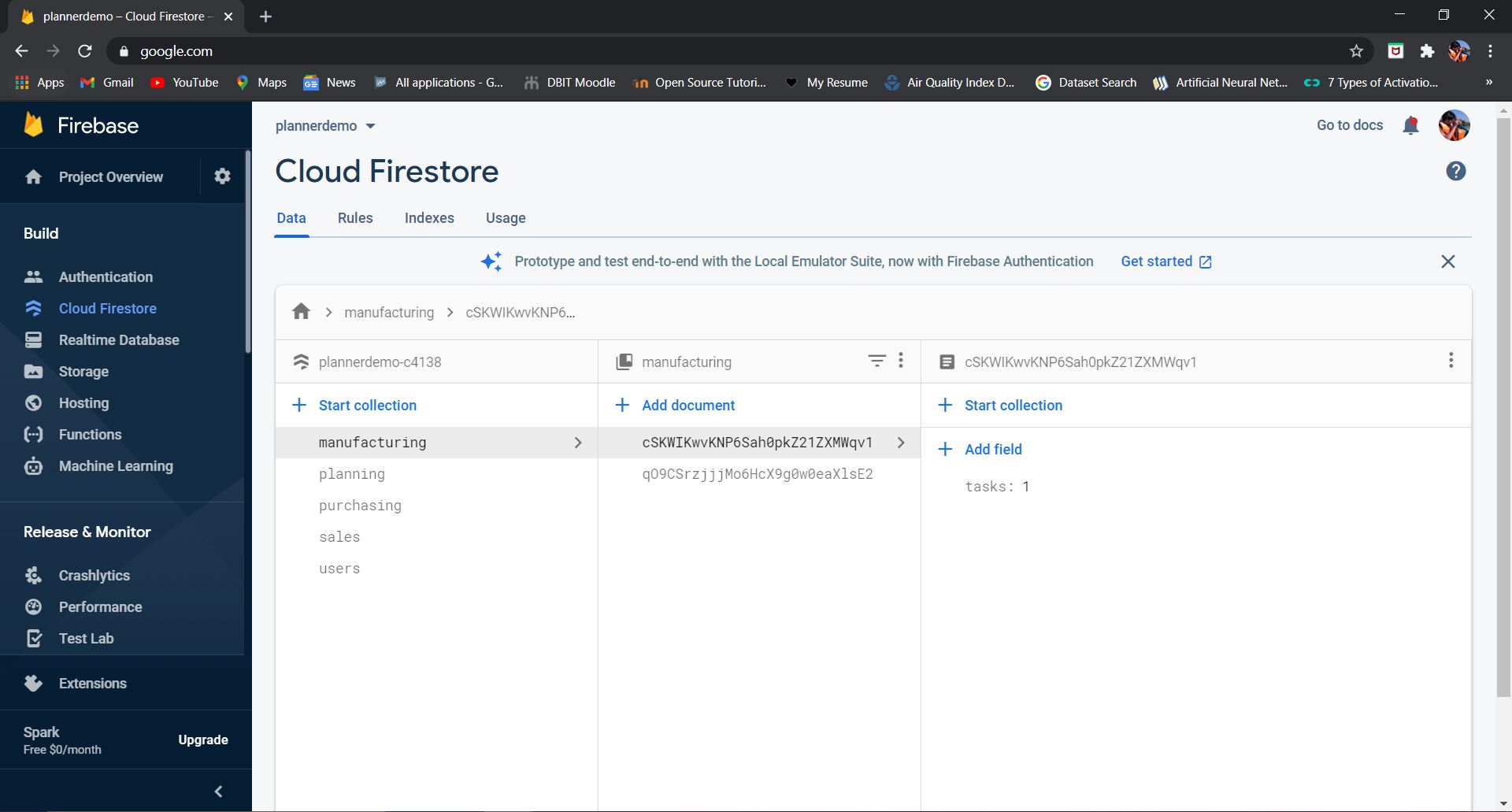
## Database states

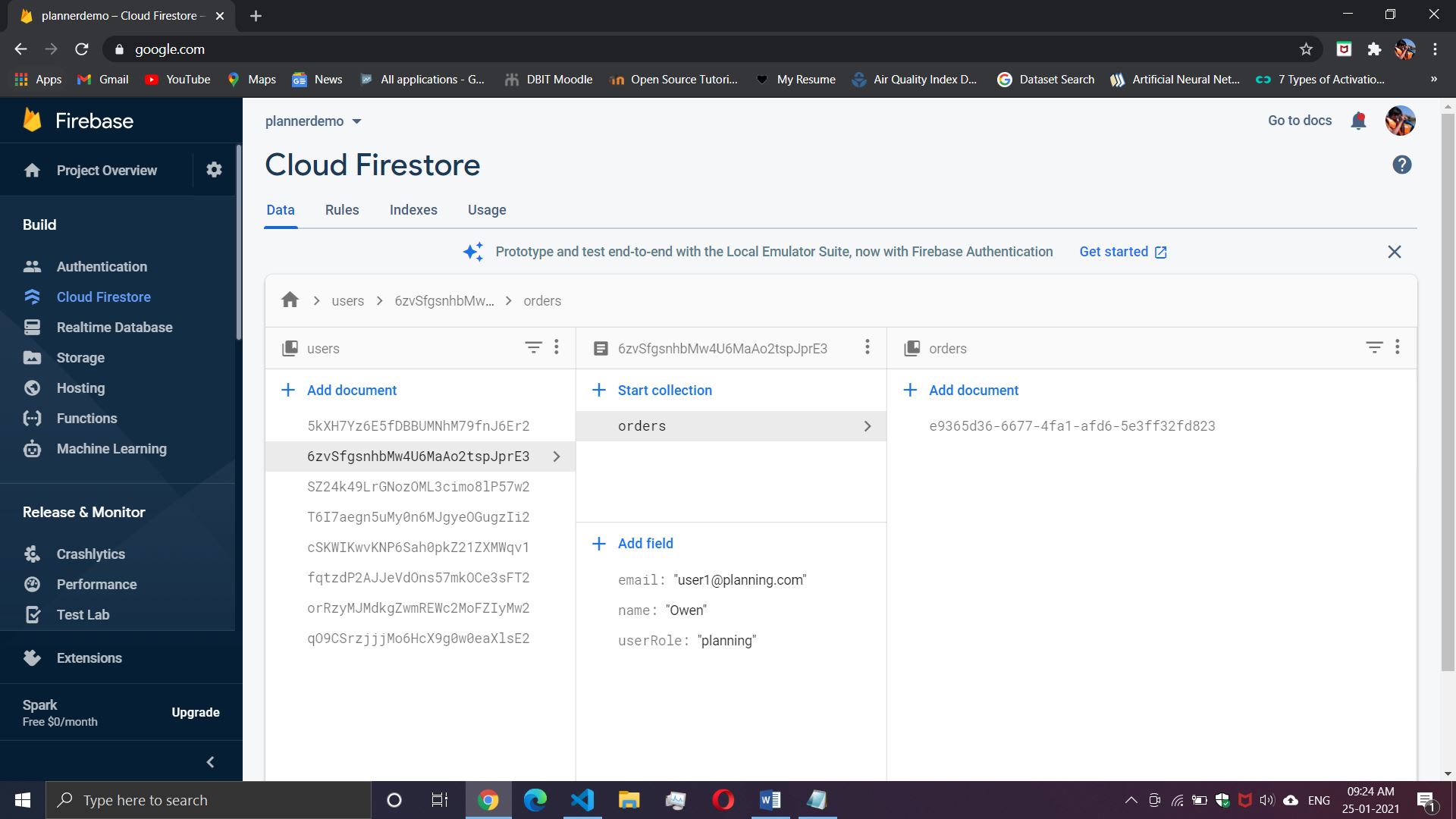
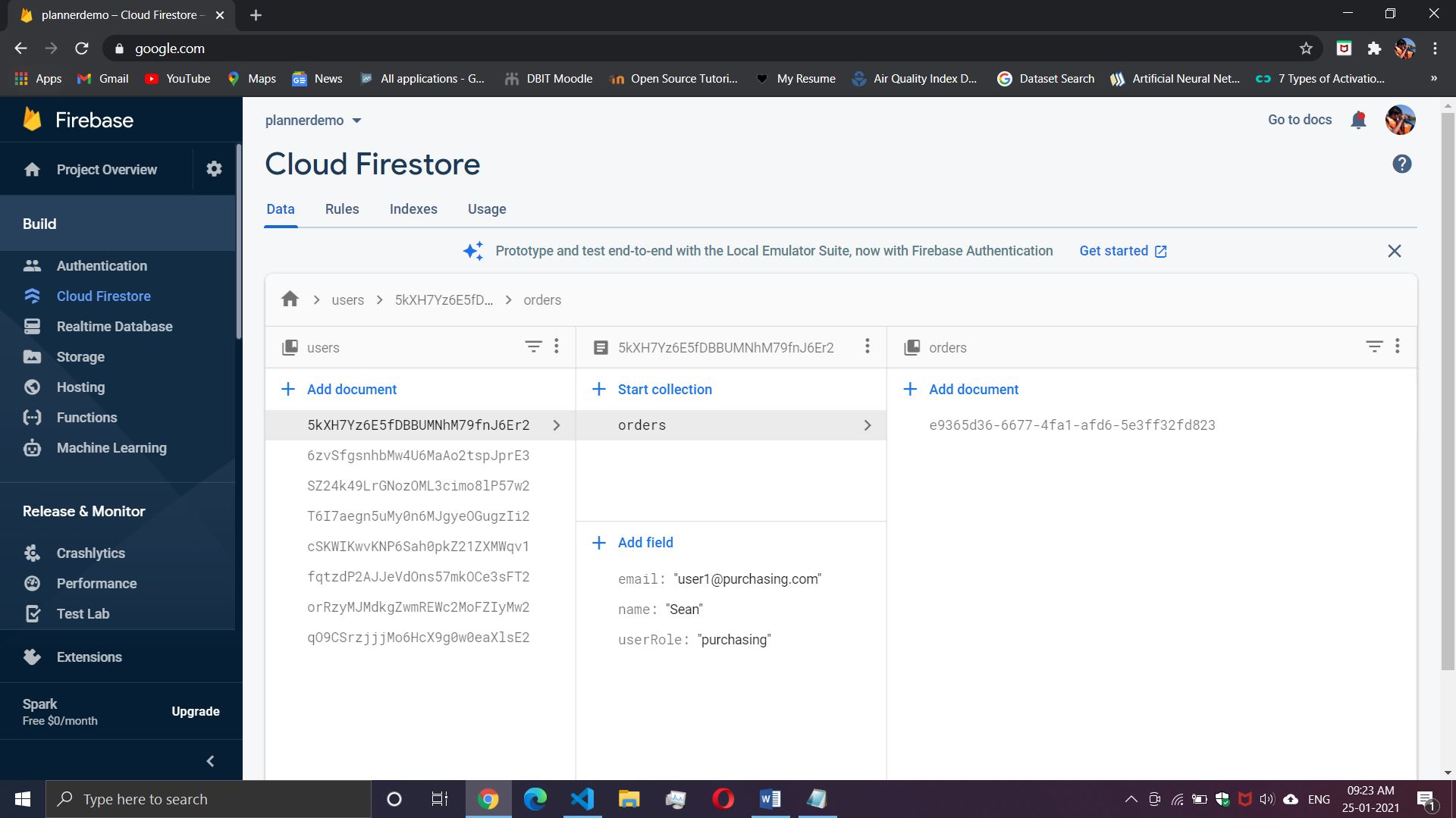
Initial State of the Database after registering the users, we can see the database initialised all the top level collections and their documents with given uid(s) appropriately as described and intended.

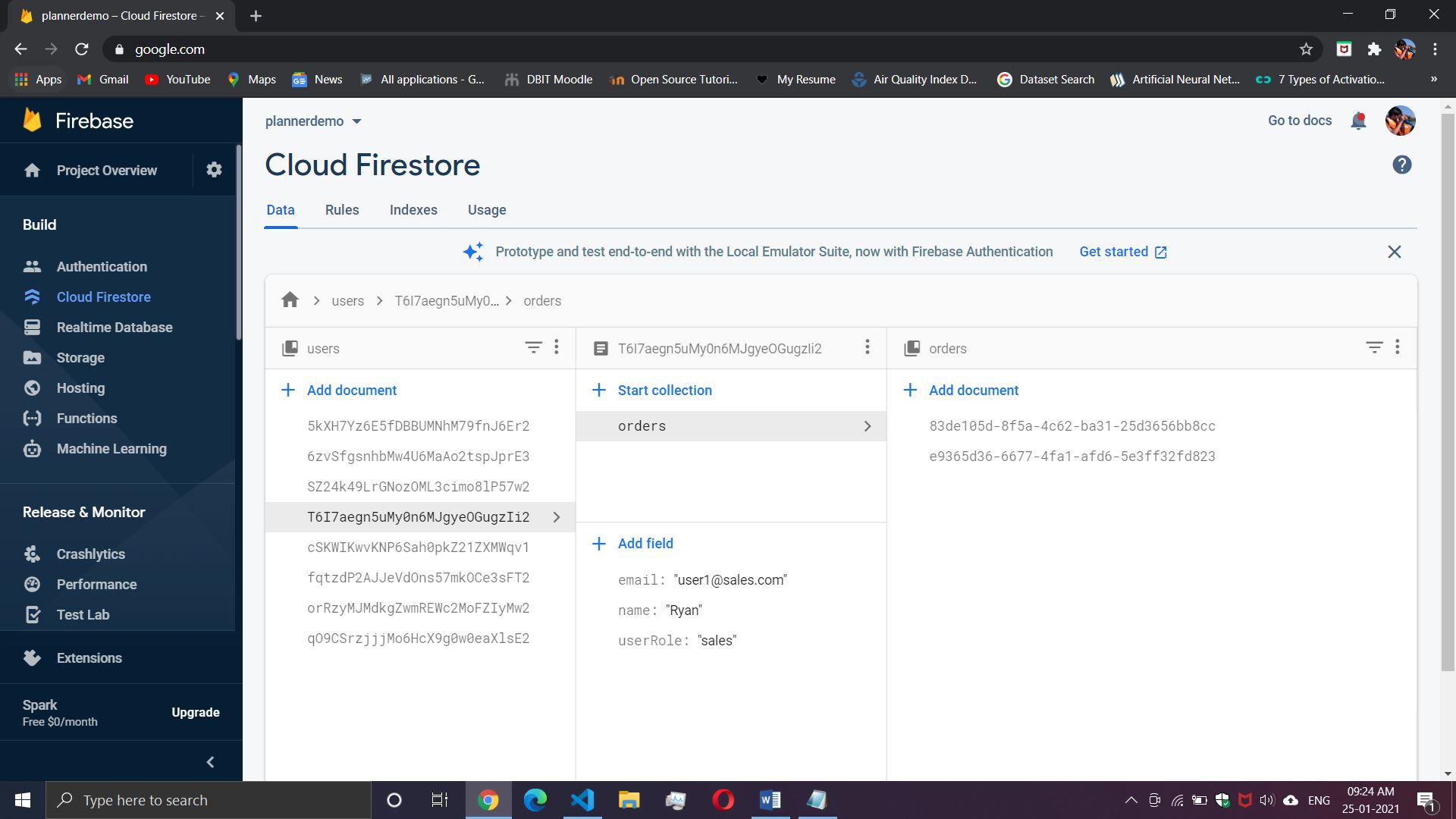


## Placing the order

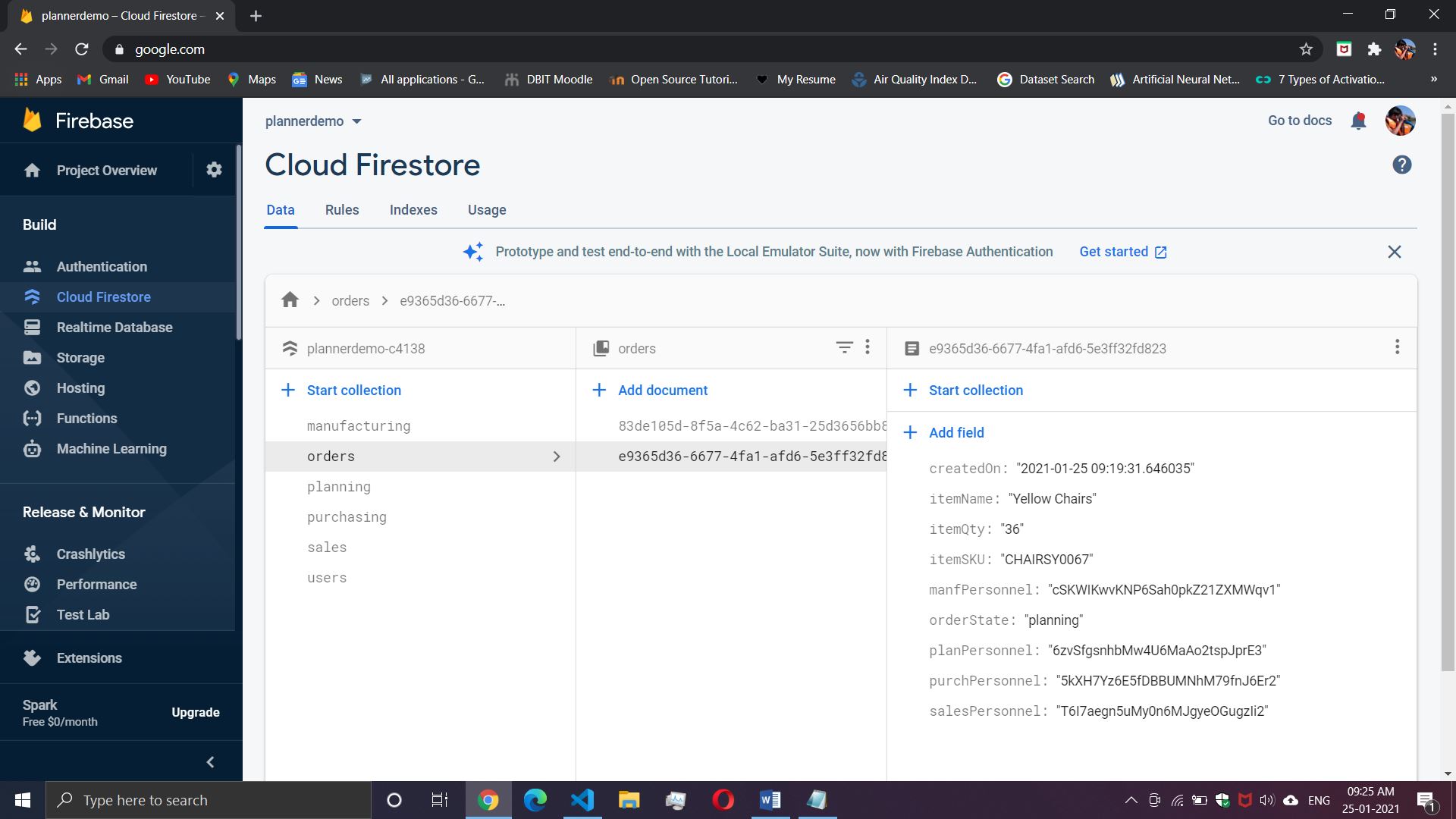
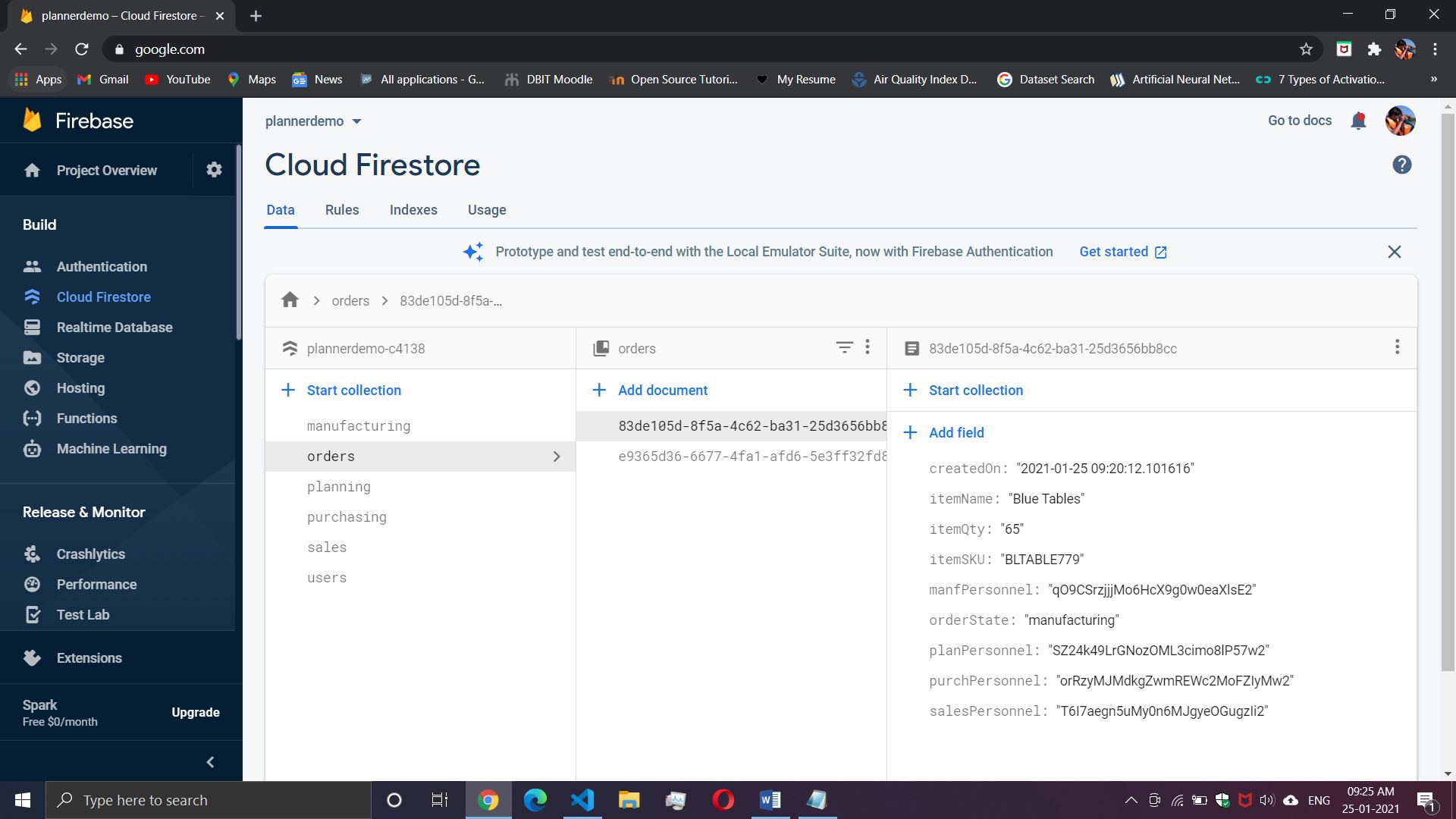
After the order is placed, it can be seen that the tasks value for the uid is updated to 1 for all the assigned users in respective roles



It can be seen below that the so called “private card list” i.e., orders collection in the uid document in the top level collection “users” is updated to have a document with the order id. It can potentially have a list of documents that the user has been assigned. This list is queried while showing the main activity.



## Below, order details are shown in the top-level orders collection by orderid



## Updated order status after the project is completed “done” and note the active tasks for the users have decremented to 0

