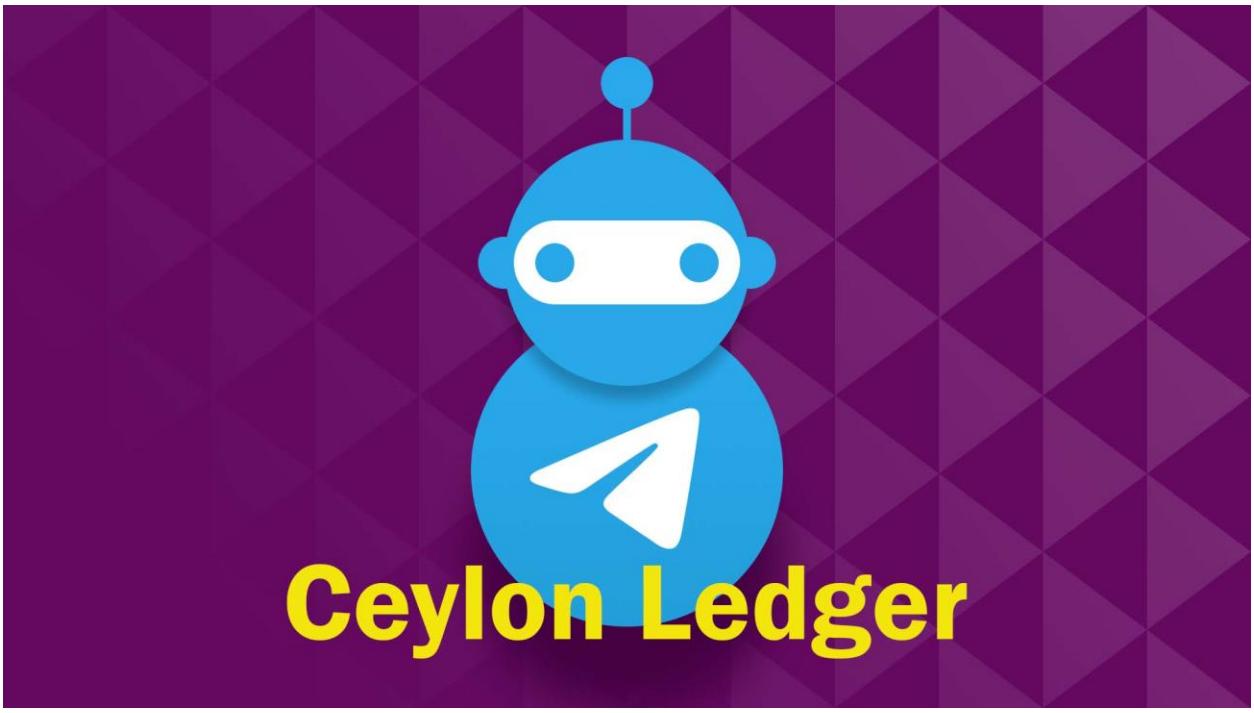
# Event Ticketing Bot for Telegram




By: Sanjaya Gunatilleke

# Table of content

**Table of content** ..... 2
**Introduction** ..... 3
  **Functionality** ..... 3
**Planning** ..... 4
  **Technical feasibility** ..... 4
  **Design** ..... 4
    Activity diagram ..... 4
    Architecture diagram ..... 5
**Implementation** ..... 6
  **Database implementation** ..... 6
  **Telegram Bot implementation** ..... 6
**Testing** ..... 10

# 1. Introduction

In order to demonstrate the capabilities of a telegram bot the author created an event ticketing telegram bot for an event. The name of the event will be “Ceylon ledger 2024”.

Key details of the Bot are given below,

## 1.1 Functionality

1. User will be able to access the bot by clicking the bot link or by scanning a QR code.

2. Provides details for the event.

3. Provides a list of commands with different types of functionality such as,
   i. /start – Gives out information about the event via poster and text message.
   ii. /help – Gives out info about the other commands and more help details.
   iii. /register – Starts the registration process storing user data to the database.

4. After the data is successfully stored the telegram bot will issue a ticket reference number and a link to join the official telegram event group.

# 2. Planning

The planning stage was divided into two types.

## Technical feasibility

- Diagram (Designing diagrams) – Draw.io
- Programming language - JavaScript and Node.js as the run time environment.
- Database - The data will be stored in JSON format in Firebase (Firestore database).
- Testing – To test the bot author will be using the desktop and mobile app of telegram.

## 2.1 Design

For the designing process the author used an activity diagram and architecture diagram.

Activity diagram

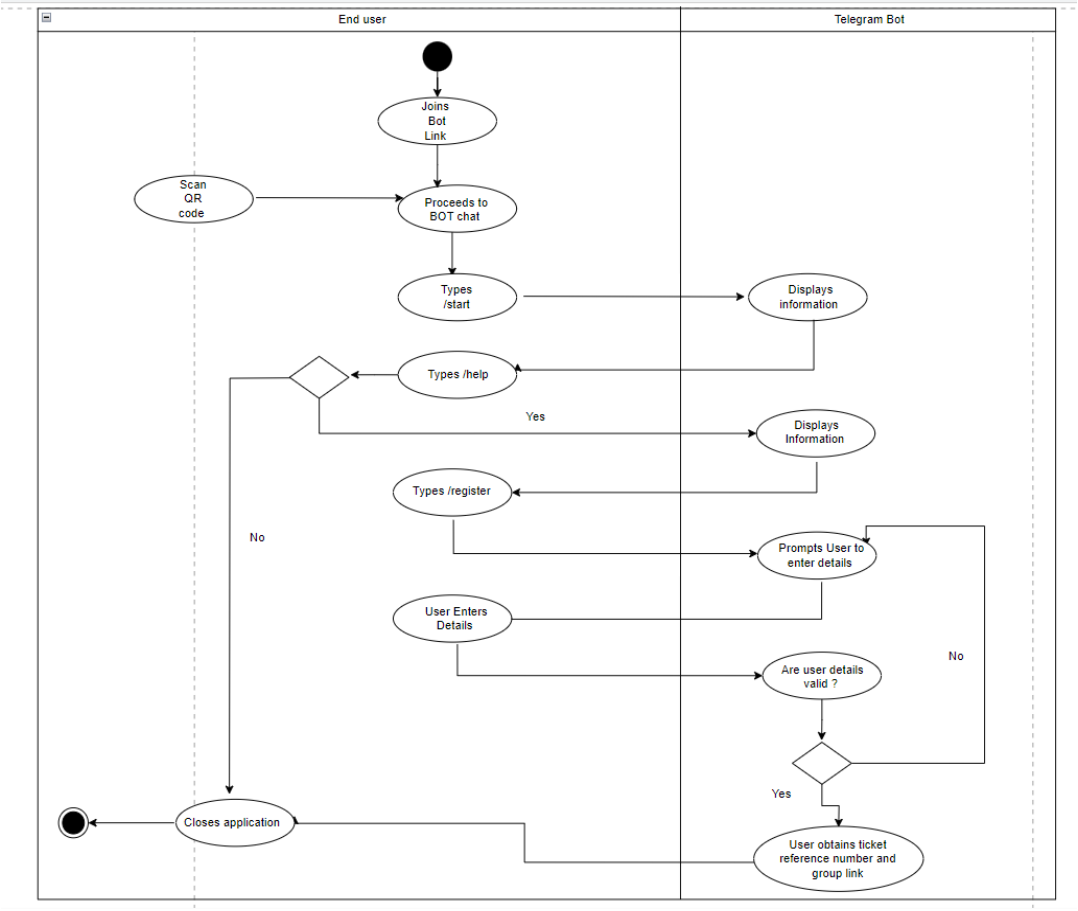



*Figure 1 : Activity diagram*

Architecture diagram

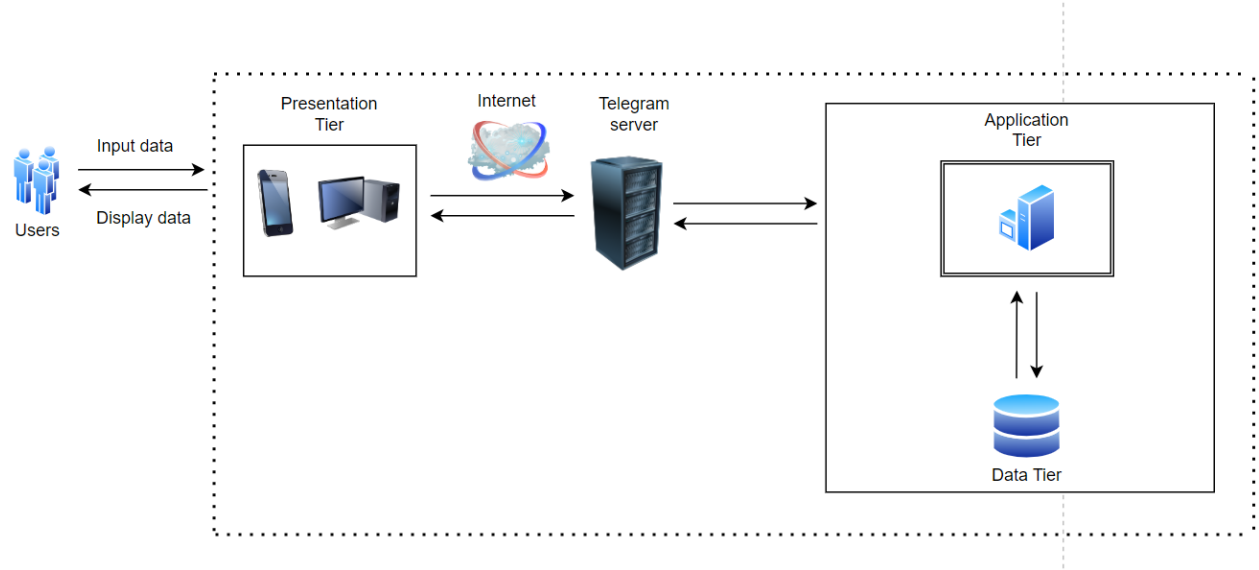



*Figure 2 : Architecture diagram*

# 3. Implementation

## 3.1 Database implementation

The database was created in Firebase's Firestore console as shown below.

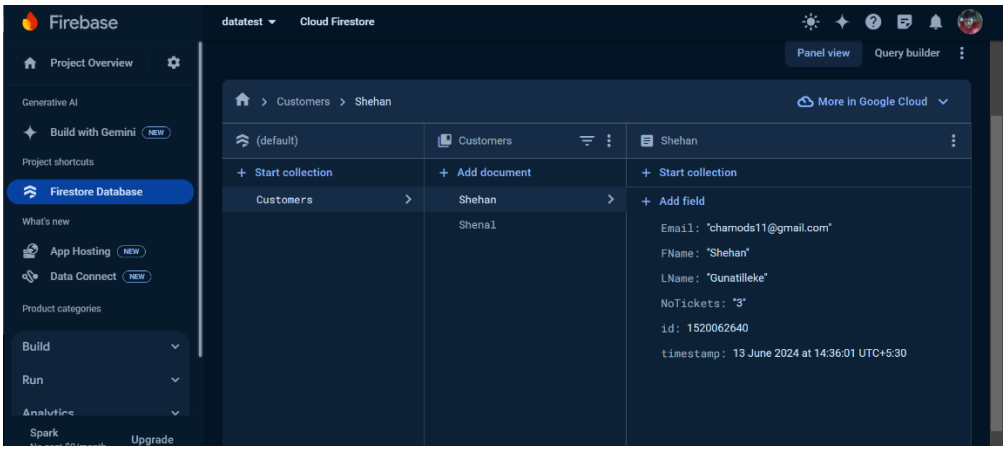



*Figure 3: Firestore database*

## 3.2 Telegram Bot implementation

The majority of the code was written inside the "structure.js" file.

The first lines are all the required modules that will be used to create the application. The main modules will be the Telegram bot module to create the Telegram bot, the Firebase admin module to interact with Firebase's Firestore database and finally a request module to make HTTP calls.

```
2   const TelegramBot = require ('node-telegram-bot-api');
3   const admin = require("firebase-admin");
4   var request = require('request');
5
```

*Figure 4 : Installed modules*

The telegram bot was made and the token was generated using Bot Father on Telegram and the generated bot token will be initialized and it's used to access the bot on Telegram. The token is imported to the "structure.js" file from the 'teletoken.js' file.

```
6   const { token } = require("./teletoken.js"); //import telegram token
7   const bot = new TelegramBot(token, {polling:true});  //initiliazes the telegram token
```

*Figure 5: Telegram bot token*

Figure 6 shows the process where the bot was created and the token was generated from BotFather on Telegram.

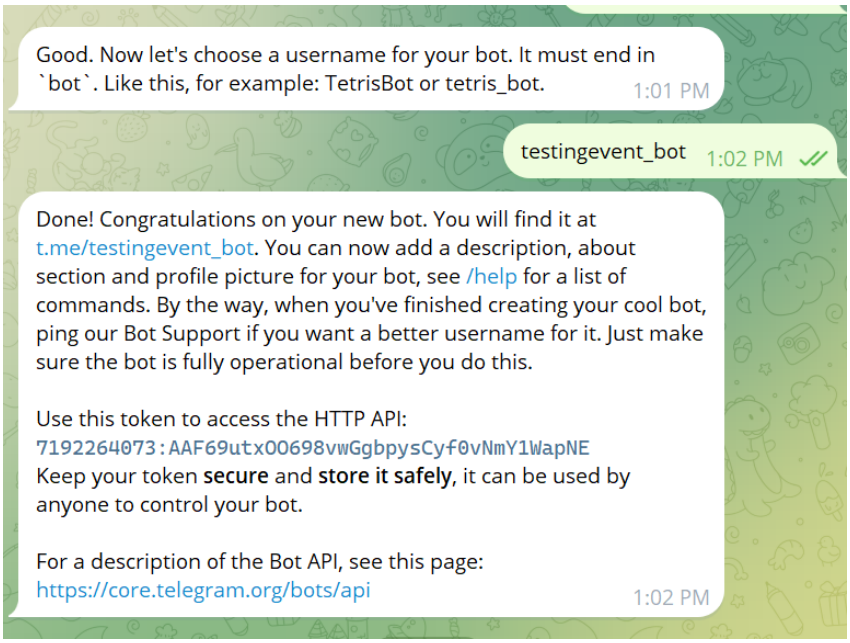



*Figure 6: BotFather token*

Next is the authentication details combined with a private key bundled in a JSON file named 'serviceAccountKey.json' that will provide permission or open a gateway for the Telegram Bot to interact with the Firestore database to access and modify data. The authentication details and private key are obtained from the Firebase console.

```
8
9   var serviceAccount = require("./serviceAccountKey.json");  //json file containing the authentication details and pr
10
```

*Figure 7: "serviceAccountKey.json code"*

The given below code section in figure 8 will initialize the Firebase admin SDK using credentials from the given private key in the 'serviceAccount' variable. The next few lines will define the database name in Firebase.

```
11  admin.initializeApp({   //obtains the firestore authenication details and private key
12    credential: admin.credential.cert(serviceAccount)
13  });
14
15  const db = admin.firestore();
16  const UsersDB = db.collection('Customers'); //Firestore database name
17
```

*Figure 8: Initializing the database*

The next few lines of code will consist of the command functions.

Figure 9 demonstrates the function for the '/start' command. The first few lines handle the message where the bot listens for the incoming messages and once the appropriate message is received it will trigger the corresponding action based on the user's input. The function will consist of many messages obtained from the telegram bot API such as "sendMesssage" which will display the reply message by the bot and "sendPhoto" which will send a photo bundled in an if-then statement. All the functions with the relevant commands follow the above rules.

```
22  //start command
23  bot.on('message', function(msg) {
24      const chatId = msg.chat.id;
25      const messageText = msg.text;
26      if((messageText).toLowerCase()== '/start') {
27        bot.sendMessage(chatId, "Welcome to the Ceylon Ledger Event. I will be your event bot.")
28        .then(() => {
29          return bot.sendMessage(chatId, "Type or touch the /help to get more info on the registration process.")
30        })
31        .then(() => {
32          return bot.sendMessage(chatId, "For more details on the event refer to the below given poster.")
33          .then(()  => {
34            return bot.sendPhoto(chatId, posterimg);
35          })
36        });
37      }
38    });
39
```

*Figure 9: /start command code*

Given below is the help command code which follows the same structure of the start command function.

```
42  //Help command
43    bot.on('message', function(msg) {
44      const chatId = msg.chat.id;
45      const messageText = msg.text;
46      {
47        if((messageText).toLowerCase()== '/help') {
48          bot.sendMessage(chatId, "Type or touch the ' /register ' command to enter personal details.")
49          .then(() => {
50            return bot.sendMessage(chatId, "To get more information regarding the event or registration process please
51          })
52          .then(() => {
53            return bot.sendMessage(chatId, "Email - chamods11@gmail.com \n \nPhone Number - 0729019141")
54          });
55        }
56      }
57    });
58
```

*Figure 10: /help command code*

As given in figure 11 the code to insert user data will consist of the users chat ID and when inserting the message it will be split into substrings using the “split()” method and then added into the database according to the given order of the message text between the user data inserted by the user.

```
// Insert User Details
bot.on('message', function(msg) {
  const chatId = msg.chat.id;
  const newMsg = msg.text.split(' ');
  const time = admin.firestore.FieldValue.serverTimestamp();
  const preregistercommand =  `To obtain free tickets insert the personal details according to the below given form
  if (newMsg[0] === '/register') {
    bot.sendMessage(chatId, preregistercommand);
  } else if (newMsg[0] === 'insert') {  //The data structure that is being entered.
    const dataId = newMsg[1].trim(); //trim removes space from sides of the string.
    const datainfo = {
      FName:newMsg[1],
      LName:newMsg[2],
      Email:newMsg[3],
      NoTickets:newMsg[4],
      id: chatId,
      timestamp:time, //Time stamp will be displayed in the firebase console
    };
```

*Figure 11 : Insert User data code part 1*

Once the data is valid it will be entered where the “dateId” constant will be collection name in the Firestore database and the “dataInfo” constant will be inserted into the Firestore database. The terminal will display a successful message as well and the bot will reply with a message containing the next few steps as shown in figure 12.

```
    UsersDB
    .doc(dataId) //adds the data into the firestore datbase
    .set(datainfo)
    .then(() => {
      console.log('Personal details stored in the firestore database:', datainfo);
        bot.sendMessage(chatId, ` ${datainfo.FName} your personal details has been successfully added.`)
        .then(() => {
          return bot.sendMessage(chatId, `Your ticket reference number is "${datainfo.id}".`)
        })
        .then(() => {
          return bot.sendMessage(chatId, "Press the https://t.me/+brzy6ZWALpk5OTY1 link to be transffered to the 'Cey
          .then(()  => {
            return bot.sendMessage(chatId, "We look forward for your participation. Thank you !");
          })
        });
    })
    .catch((error) => {  //If an error is encontured.
      console.error(' Error storing facts in Firestone:', error);
      bot.sendMessage(chatId, 'Details have not been added. Please check the given format and for more assitance use
    });
    }
  });
```

*Figure 12 : Insert user data code part 2*

# 4. Testing

## 4.1 Setting the event ticketing telegram bot

In order to run the project the follow the given steps.

1. Obtain all the files from the GitHub link ‘ https://github.com/chamodsprog/CeyLabs-Intern-TGBot-EventRSVP ’

2. Install each necessary modules one by one in order using the terminal,

```
npm init
npm i node-telegram-bot-api
npm i firebase-admin
npm i request
```

3. After the installation run the project by typing the below code in the terminal.

```
node structure.js
```

## 4.2 Testing the telegram bot

For the testing process the mobile app of telegram will be used. Each step the user will use is given below using screenshots.

1. First the user will scan the given QR code and joins the group.


*Figure 13: Poster for QR code scan*

The user can also join the telegram bot by accessing the URL ‘ t.me/testingevent_bot ’.

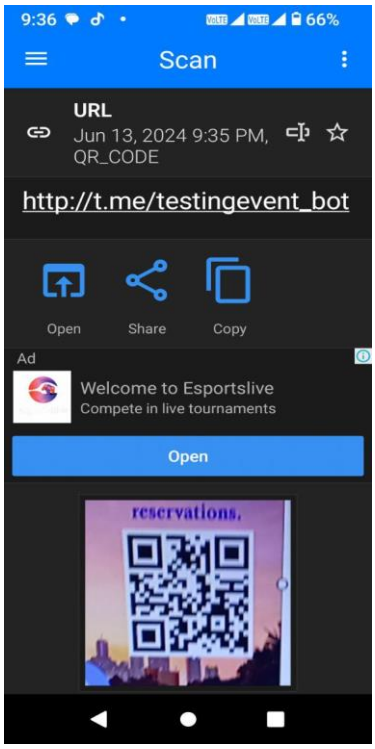



*Figure 14: The link generated once the QR code was scanned.*


*Figure 15: New user entering the group*

2. User gets greeted by the '/start' command.

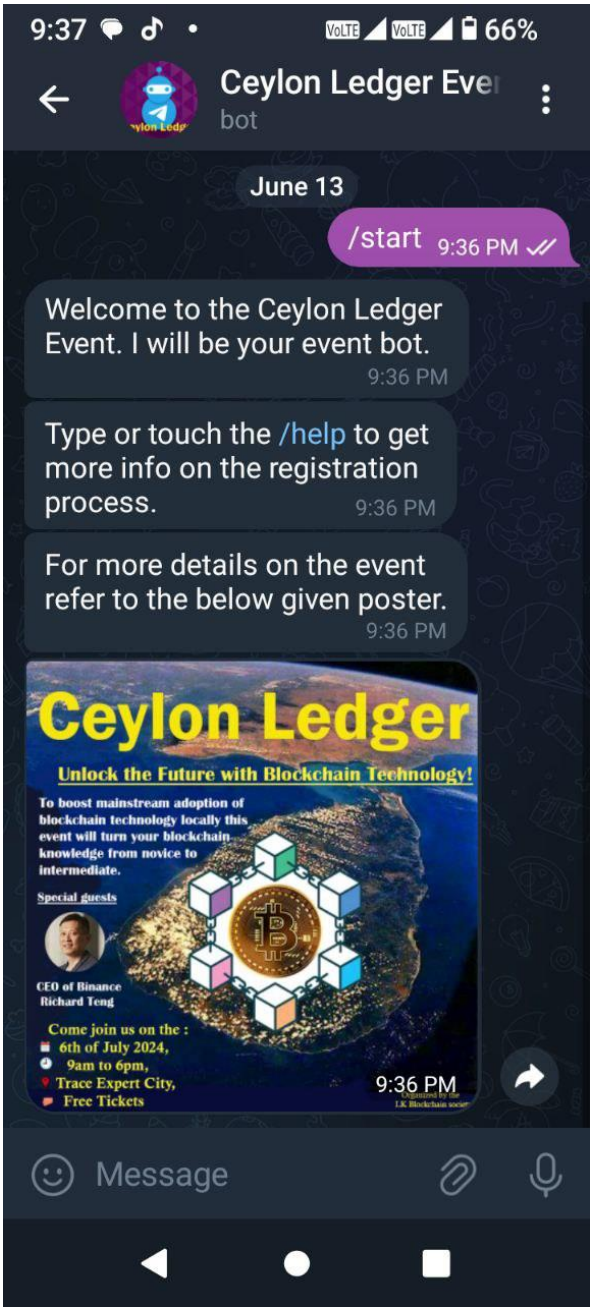



*Figure 16: Testing procedure for /start command*

3. User utilizes ‘/help’ command.

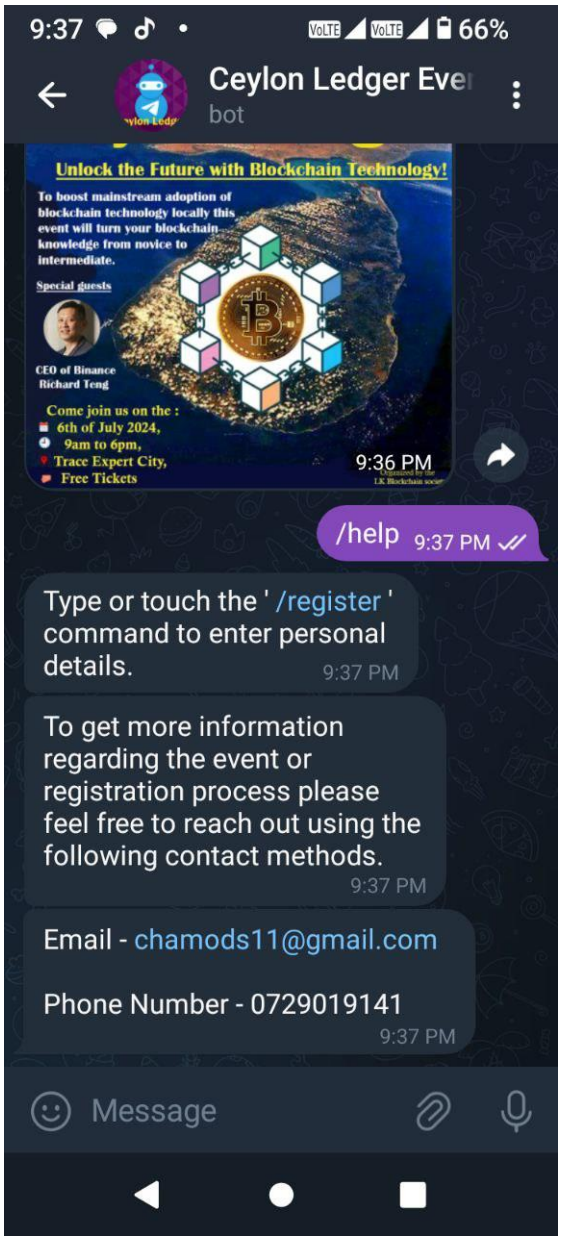



*Figure 17: Testing process for /help command*

4. User utilizes ‘/register’ command and inserts the personal data.

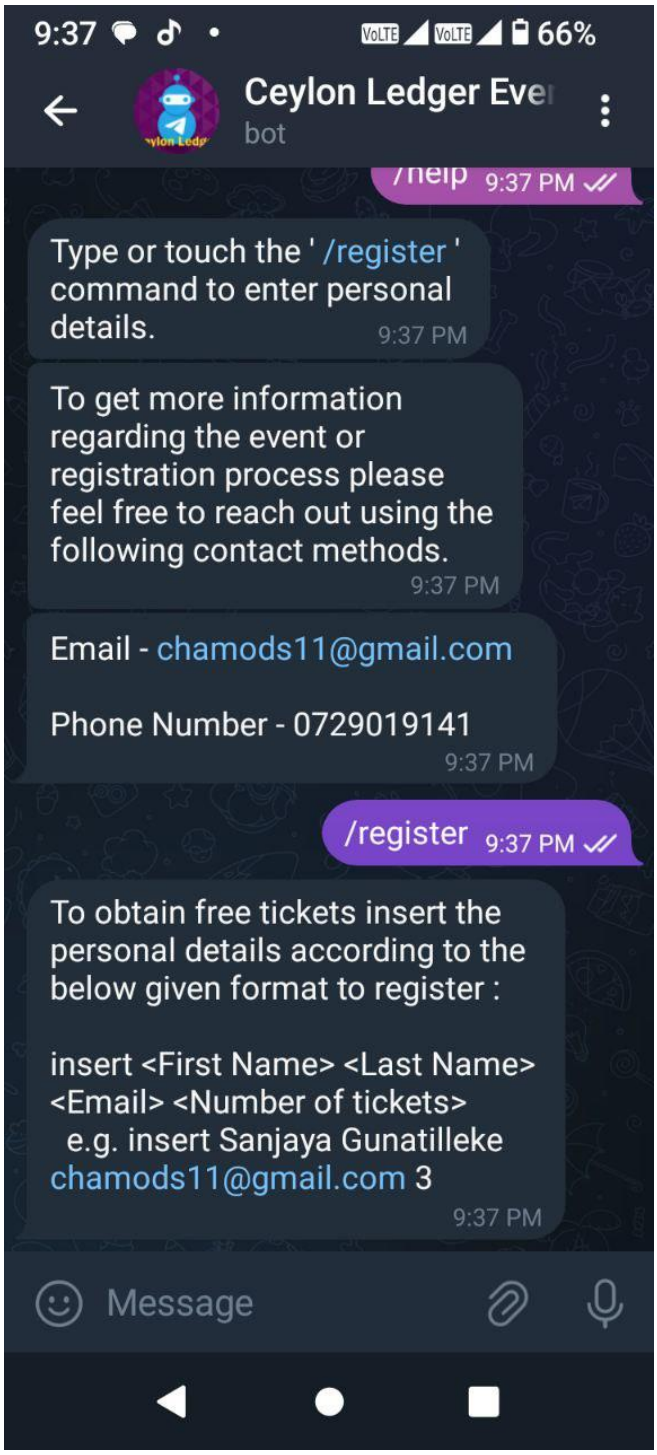



*Figure 18 : Testing process for /register command*

The user successfully inserts the data using the given format.


*Figure 19: Testing process for the insert command screenshot 1*


*Figure 20: Testing process for the insert command screenshot 2*

The inserted details are displayed in the Firestore database along with the time stamp.

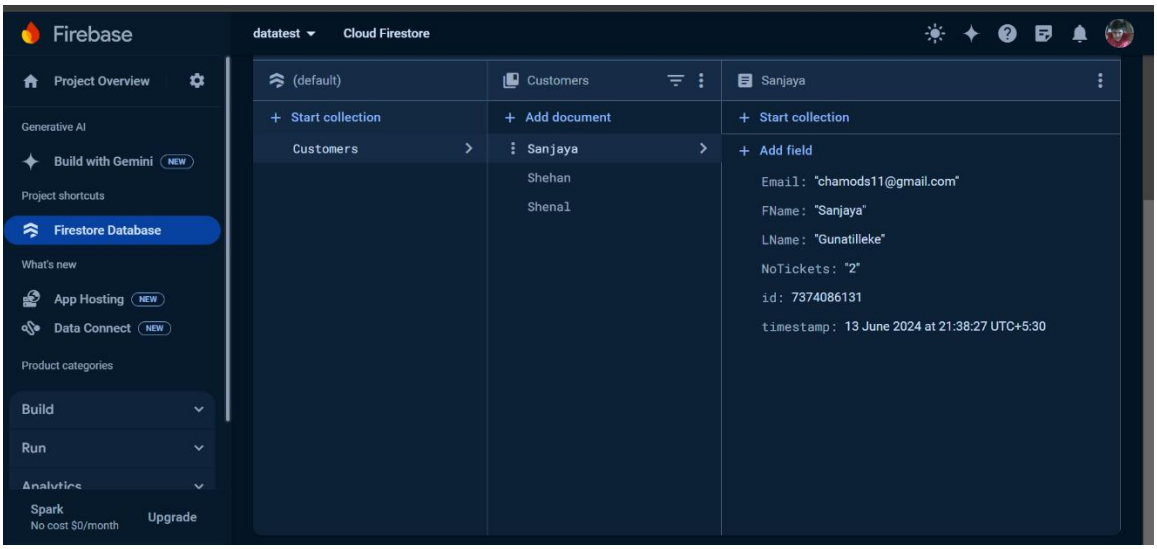



*Figure 21 : The inserted details in the database*

The console log message in the terminal after inserting the user inserts the details.

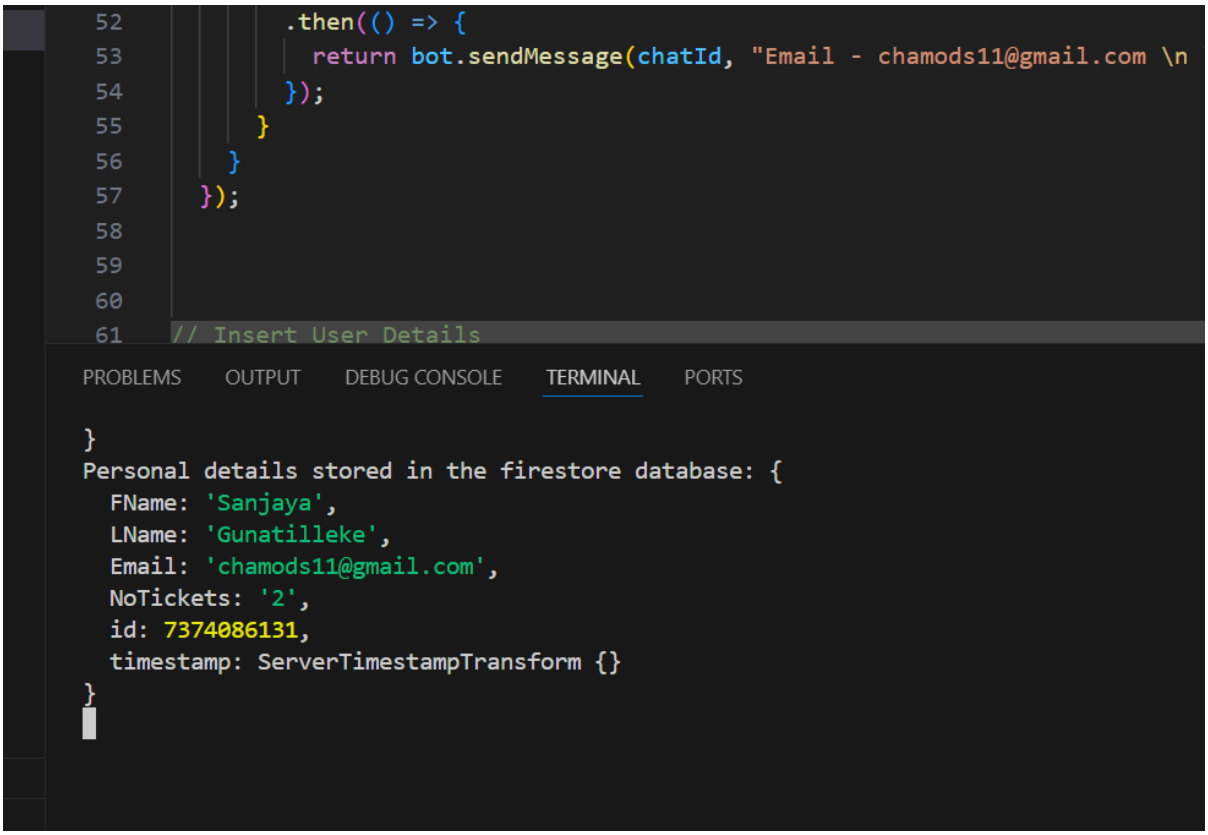



*Figure 22: Terminal message after the user inserts the data*

# 5. Conclusion

## 5.1 Future Improvements

Future improvements that could be made to further improve the functionality of the telegram bot.

1. Introduction of data validation.
   Implementing data validation points can help reduce errors and inconsistencies in the database and provide a seamless experience for the user. The user can fill the user details message by message on telegram thus making it easier for data validation.

2. Encapsulation
   To avoid data breaches and strengthen security data encapsulation can be used. Due to the project having private keys, tokens and authentication details, it is vital to introduce data encapsulation.

3. Better coding best practices