This is the requirements for our api

1. **createSignal**

   Description:
   This function will creat a document in the “signals” collection.
   The data for this documents will received from the params.
   After the data is stored successfully a push notification will be sent to the users with the right channel.

   This function will be open only for the admin, so its important to creat set of rules so only the admin could send them

   Params:

   coinId - String
   The coin id from CoinGecko
   https://www.coingecko.com/en/api/documentation

   risk - Int
   An ENUM with the following :
   HIGH_RISK(100), MEDIUM_RISK(101), LOW_RISK(102)

   scalp- Int
   An Enum with the following:
   SCALPING(200), SWING(201), HOLD(202)

   stop - Double

   targetList - a list of targets
   Target:
   TargetPrice - Double
   TargetPercentage -Double

   SignalTimestamp - timestamp
   https://firebase.google.com/docs/reference/node/firebase.firestore.Timestamp
   This param will be created by the function at the time of the creation of the document

2. **getAllSignals**

Description:

This function will fetch data from signals collection based on the params.
Notice - we are using paganation.
Notice 2 - we are using CoinGecko Api to get the latest coin price

This function will is open for all users, notice that if the user is not registered and if his **subscriptionType==FREE(102)** then the data is different

Params:

limit - Int
For paging purpose - the client can control the amount of documents limit.

Default is 20

lastVisibleDocumentId:Int
For paging purpose - this will be the last visible document the client as seen, so the function must first fetch that document and then used it an a query
https://firebase.google.com/docs/firestore/query-data/query-cursors#node.js_3
Default is null

Response:

coinId - String
The coin id from CoinGecko
https://www.coingecko.com/en/api/documentation

coinPrice- Long
The coin price from CoinGecko
Price in USD
Here is a simple api request snippet:
https://api.coingecko.com/api/v3/simple/price?ids=bitcoin,lightcoin&vs_currencies=usd

subscriptionType - Int
An ENUM with the following :
SUBSCRIBED(100), FREE_TRAIL(101), FREE(102)

risk - Int
An ENUM with the following :
HIGH_RISK(100), MEDIUM_RISK(101), LOW_RISK(102)

scalp- Int
An Enum with the following:
SCALPING_LONG(200), SCALPING_SHORT(201), SWING_LONG(202)
, SWING_SHORT(203), HOLD(204)

stop - Double

targetList - a list of targets
Target:
TargetPrice - Double
TargetPercentage -Double

signalTimestamp - timestamp
https://firebase.google.com/docs/reference/node/firebase.firestore.Timestamp
This param will be created by the function at the time of the creation of the document

In order to make the client side an"easy life" I think the following approach could work

After retrieving the documents, using the pagination, make a food loop
And on them and build the return list (the returned list will not look the same as the fetched collection)

Then - map the used coins id in the parsed documents.

Run another food loop on the parsed collection, this time add the coin value to them.

Send back the data.

Notice:
If the use **subscriptionType==FREE(102)** do not return a a real data other then the coin price

3. **RegisterUser**

   Description:

   This function will create a new document inside the users collection

   No rules needed in this function

   Params:

   userName - String

   email - String

   creationTimestamp - timestamp
   https://firebase.google.com/docs/reference/node/firebase.firestore.Timestamp
   This param will be created by the function at the time of the creation of the document

   acceptedTermsTimestamp - timestamp

   fcmToken:String
   Push token - must be stored so we could send chat messages

   os :Int
   The Operating system “android” , “ios”

4. **getGuides**

   Description:

   No rules needed in this function

   This function will fetch all the collection in “Guides” collection
   We are going to manually add data to this collection, all the function has to do in this point is to fetch it and parsed it with more readable response

5. **getStates**

Description:

No rules needed in this function

This function will fetch all the collection in “States” collection
We are going to manually add data to this collection, all the function has to do in this point is to fetch it and parsed it with more readable response

6. **getChatMessagesWithAdmin**

Description:

This function will fetch data from “chat” collection based on the params.
Notice - we are using paganation.

This function will is open for all users

Params:

limit - Int
For paging purpose - the client can control the amount of documents limit.

Default is 20

lastVisibleDocumentId:Int
For paging purpose - this will be the last visible document the client as seen, so the function must first fetch that document and then used it an a query
https://firebase.google.com/docs/firestore/query-data/query-cursors#node.js_3
Default is null

Response:

message - String

messageTimestamp - timestamp
https://firebase.google.com/docs/reference/node/firebase.firestore.Timestamp
This param will be created by the function at the time of the creation of the document

creatorUserId:String
The user id who wrote the message

receivedUserId:String
The user id who received the message

Notice:
The chat message at this point is very simple - just a message and a timestamp
However - it muse be structure very well
So we want to keep each chat dialog in a nested collection inside “adminChat” collection.
Every nested collection will be have his owen unique id based on a random UDID
And the dialog (the chat between the users) will be stored inside the user document
user->dialogs->{nested collection} -> dialog(“dialog id”){“last message”:String, “lastMessageTimeStamp”:timestamp}

7. **sendChatMessage**

Description:

This function will create a message collectition inside “adminChat” collection.
If the Dialog (chat between 2 users) is not exists, create a new dialog with random UDID, then save that udid inside the user document as described above.

This function will is open for all users

After the data is stored successfully a push notification will be sent to the receiver User.

Params:

message - String

messageTimestamp - timestamp

creatorUserId:String
The user id who wrote the message

receivedUserId:String
The user id who received the message

Notice: Admin user will have a Boolean “isAdmin:true”