Firestore Security Rules with Firebase Auth Based on Document and Fields

Mehmet Yaz
Dec 20, 2019 · 5 min read ★

Unlike recent past, it is more important to store information safely ,not distributing information as much as possible (As I tried to project in the cover photo). In this case data security is one of the most important problems today. .

Firestore is a document oriantted NoSQL database that very useful and scalable. And it has a fairly easy security infrastructure.

You can use Firebase auth services in Firestore security rules infrastructure. So it's easy to create user-based security rules.

Unfortunately, I didn't get enough documentation on this subject, so I felt the need to write this article.

After this information, which is already known to everyone, I will give a quick example.

Table of Contents

1) Understanding Firestore Security Rules

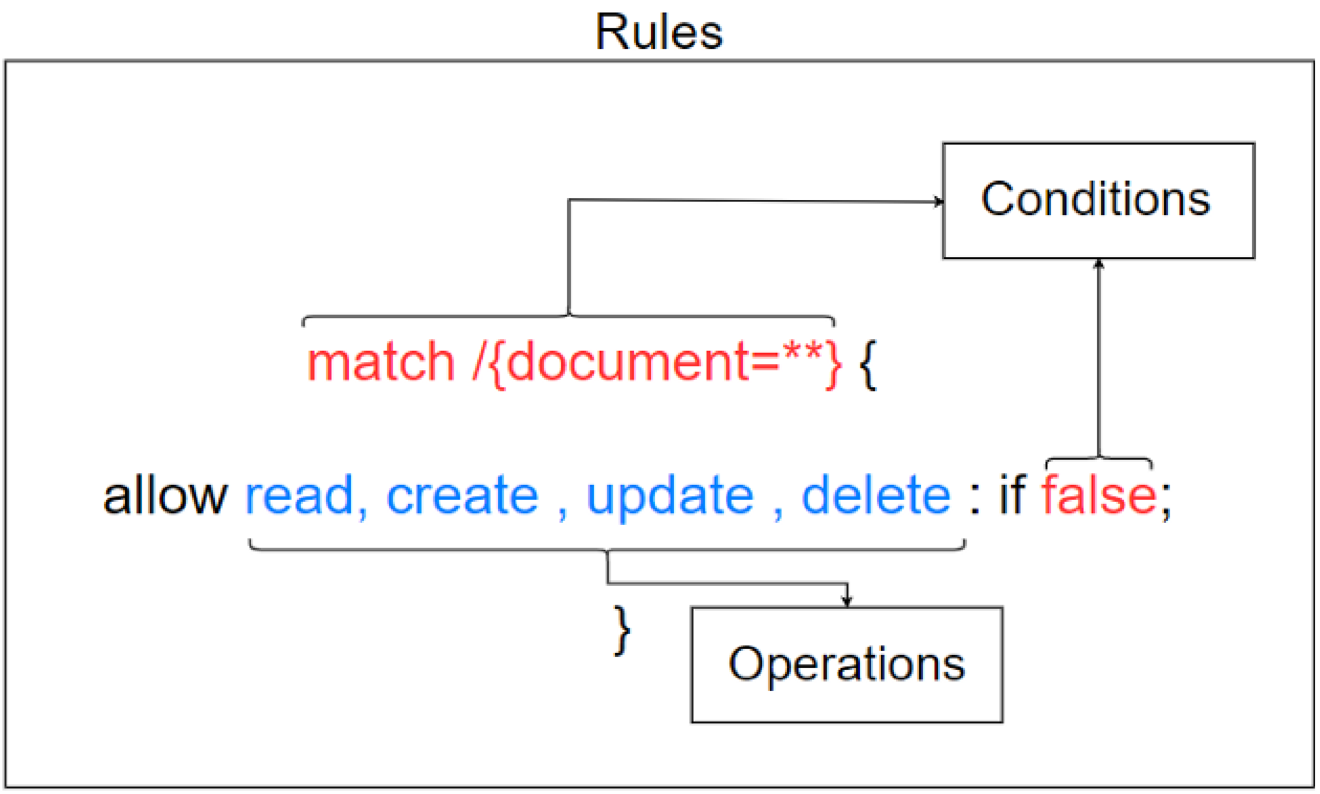
Cloud Firestore Security Rules work explained on the picture. There are 4 king of operation and each rules have 2 condition.

Each rules have ;

A) `match /path` conditon. You can write rules for each document or collection.

B) Operations list to be allowed.There are 4+1 kind of operations ; `Read` , `Delete` , `Update` , `Create` (or “ `write` ” valid for `delet` , `update` , `create`). You can write if condition each opearetion seperate.

C) If condition . where the first condition is satisfied, the requested operation shall be permitted.



2) Syntax

```
match /{document=**} {
  allow read, write: if false;
}
match /users/{userId} {
  allow read : if true;
  allow delete , update : if request.auth.uid == userId;
  allow create : if true;
}
```

If you use document name include “ `{}` ” you can use parameter that column. Also this match condition it applies to all documents at that level. (for example “match all users” in picture second match).

Operators

`&&` and

`||` or

`==` is equal

`!=` is not equal

`request` request side informations

`resource` database side informations

`request.auth` request side auth informations

`request.resource` request side parameters

`request.resource.data` change request data

`resource.data` resource(in firestore) data

You can get data with `.fieldName` both side data.

3) Document Basis Rules

```
 match /{document=**} {
allow read, write: if false;
}
```

All files are closed to all operations unless there are exceptions. It is recommended that you add this column at the beginning of the rules and write a rule for each permission.

```
 match /infos/about {
allow read : if true;
}
```

Everyone `read` about document but not delete, create or update.

```
 match /books/{bookId} {
allow read , create: if true;
}
```

Everyone `read` , and `create` , but anyone doesnt `update` , `delete` any document in a books collections.

4) Document Rules with Firebase Auth

```
 match /users/{userId} {

allow read ,delete , update : if request.auth.uid == userId;

allow create : if true;

}
```

Everyone `create` document in a users collection but only users whose userId is equal to the documentID can do other operations

```
 match /users/{userId}/publishedPosts/{postId} {

allow read : true;

allow create : if request.auth.uid == userId;

allow delete : if false;

}
```

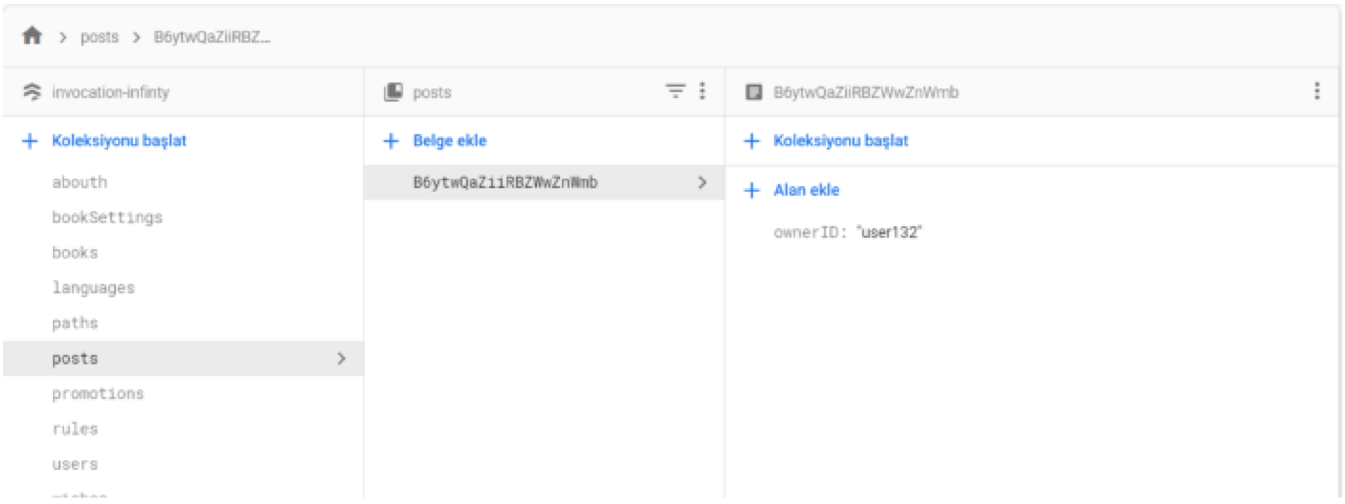
Everyone `read` this user's all published posts. But only this user `create` own post and anyone does not `delete` post.

In this case, no one is allowed to `update` .

5) Field Based Rules

```
 match /posts/{postID}{

allow read : true;

allow write: if request.resource.data.ownerId == request.auth.uid;

}
```

Everyone can `read` post but only this user can `update` , `delete` and `create` . This example only " `user132` " can write operations.


posts > B6ytwQaZiiRBZ...

invocation-infinty

Koleksiyonu başlat

abouth
bookSettings
books
languages
paths
posts
promotions
rules
users

posts

Belge ekle

B6ytwQaZiiRBZWwZnWmb

B6ytwQaZiiRBZWwZnWmb

Koleksiyonu başlat

Alan ekle

ownerID: "user132"

wishes

Other example :

```
 match /users/{userId} {

allow read ,delete , update : if request.auth.uid == userId;

allow create : if request.resource.data.age > 18;

}
```

We have added another condition to the example in the fourth article : Everyone can create document if field of "age" is geater than 18 .

6) Optimization

We need to optimize our database architecture according to rules. Firestore's recommendations include information that we need to store confidential information in sub-collections and process it with cloud functions. This is a good solution. But not cheap everytime.

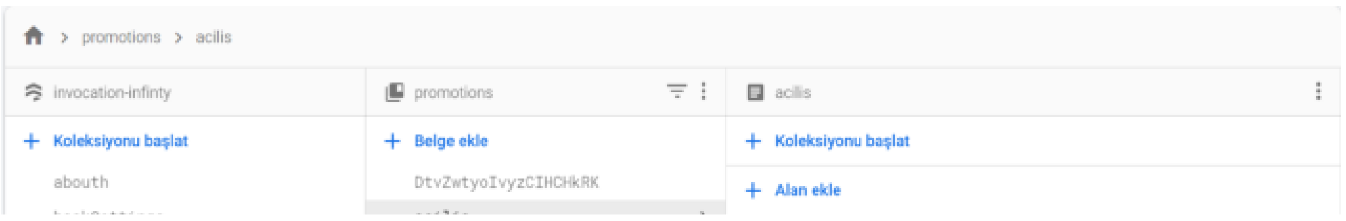
We may store the information of the owner of an anonymous post in sub-collections.This information is written once and is often unreadable. (Because it is anonymous.)

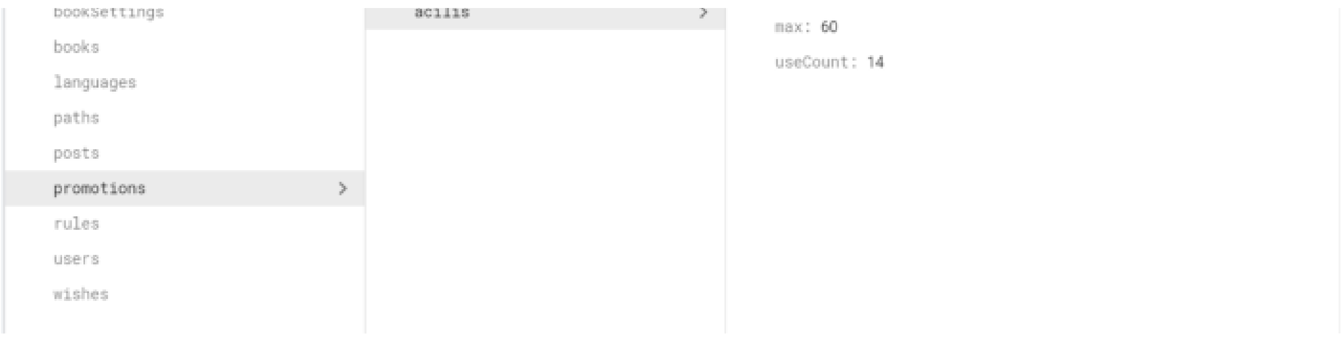
But for example we have a promotion. A document for each promotional code. Suppose the promotion has a number of uses and a quota. Certainly, malicious people will want to change these numbers. So we have a document that everyone can read but not everyone can change. According to Firestore's suggestion, write your promotional uses elsewhere and process with functions. This is mean cost for +2 read +2 write operatins. But it will often be costly to do this for every small data.

For these reasons, I will pass the rules on a field basis.

My Solution;

7) Complex Field Based Rules



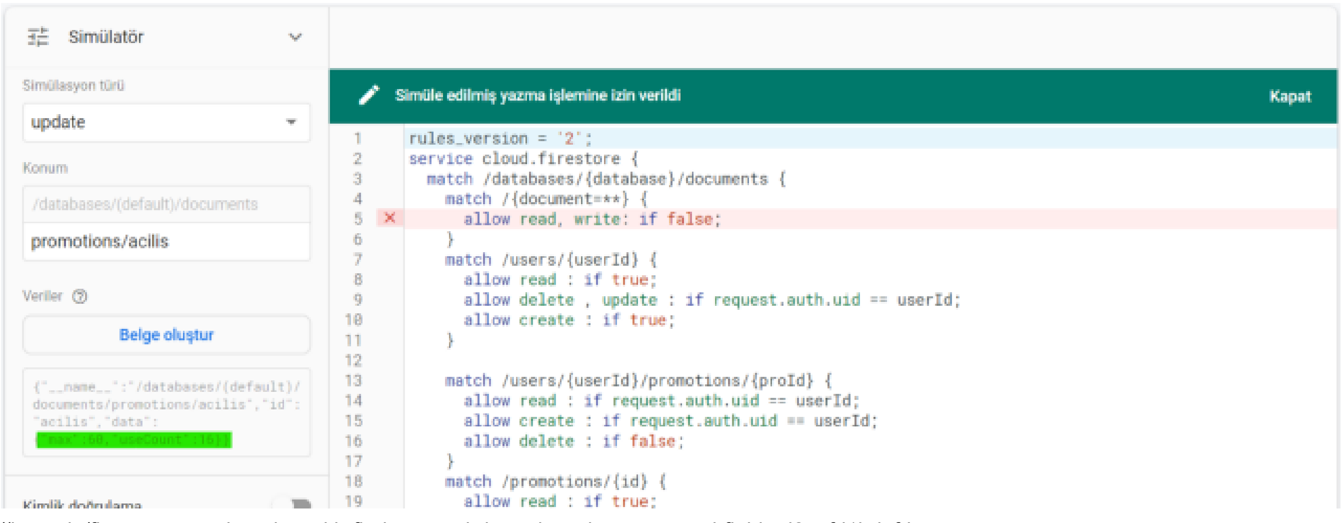


This is my promotion document.

```
 match /promotions/{id} {
allow read : if true;
allow update : if resource.data.useCount < request.resource.data.useCount
&& resource.data.max == request.resource.data.max;
}

 match /users/{userId}/promotions/{proId} {
allow read : if request.auth.uid == userId;
allow create : if request.auth.uid == userId;
allow delete : if false;
}
```

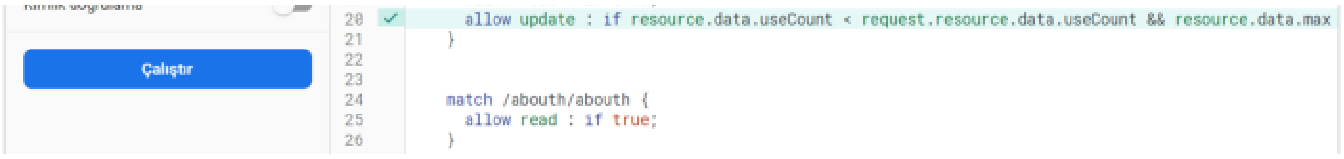
In this case. Everyone can `read` promotion document in a promotions collections. Anyone cant `delete` and `create` .

Also IF `useCount field value in incoming data is greater than in value of original data` (only increment) and `max is equal to original data` (not permission update `max` field)people can `update` document.



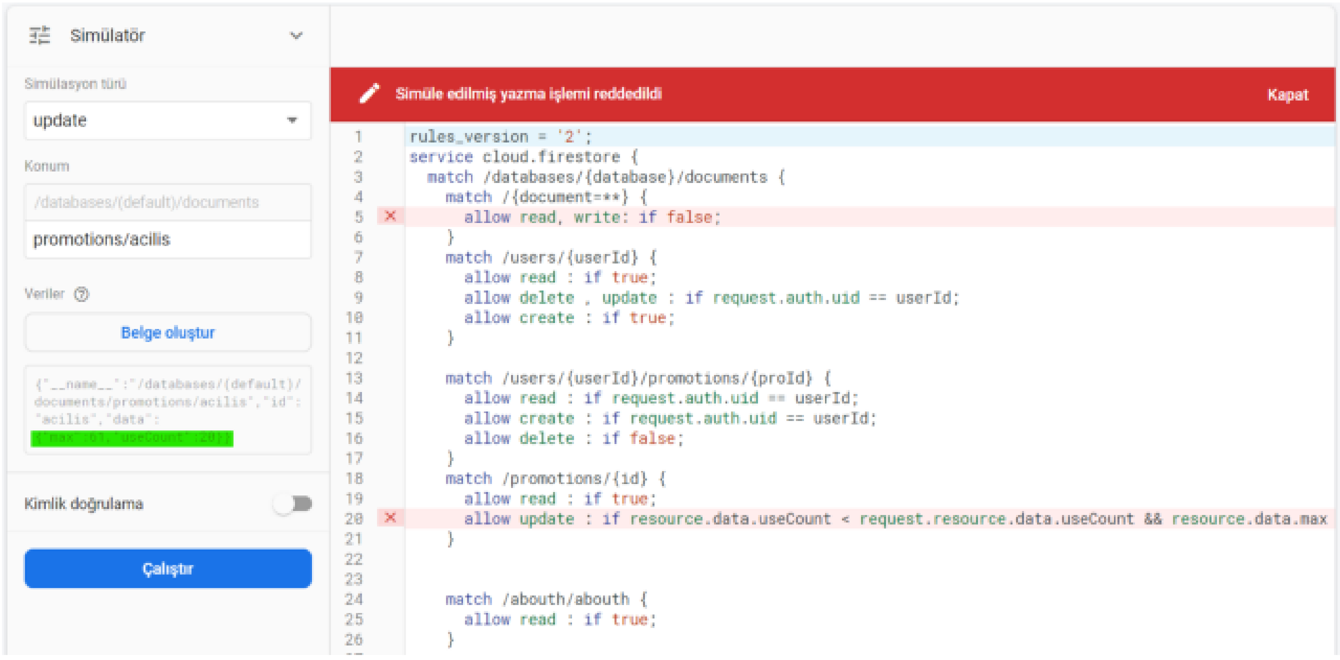
```
{"__name__":"/databases/(default)/documents/promotions/acilis","id":"acilis","data":{"max":60,"useCount":16}}
```

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if false;
    }
    match /users/{userId} {
      allow read : if true;
      allow delete , update : if request.auth.uid == userId;
      allow create : if true;
    }

    match /users/{userId}/promotions/{proId} {
      allow read : if request.auth.uid == userId;
      allow create : if request.auth.uid == userId;
      allow delete : if false;
    }
    match /promotions/{id} {
      allow read : if true;
```



```
      allow update : if resource.data.useCount < request.resource.data.useCount && resource.data.max
    }


    match /abouth/abouth {
      allow read : if true;
    }
```

Please read green highlighted.



update

/databases/(default)/documents
promotions/acilis

```
{"__name__":"/databases/(default)/documents/promotions/acilis","id":"acilis","data":{"max":61,"useCount":20}}
```

Simüle edilmiş yazma işlemi reddedildi

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if false;
    }
    match /users/{userId} {
      allow read : if true;
      allow delete , update : if request.auth.uid == userId;
      allow create : if true;
    }

    match /users/{userId}/promotions/{proId} {
      allow read : if request.auth.uid == userId;
      allow create : if request.auth.uid == userId;
      allow delete : if false;
    }
    match /promotions/{id} {
      allow read : if true;
      allow update : if resource.data.useCount < request.resource.data.useCount && resource.data.max
    }


    match /abouth/abouth {
      allow read : if true;
    }
```

Permission denied. Because max is different than original data.

Other Example:

```
match /posts/{postID} {

allow read: if true;

allow create: if true;

allow update: if request.resource.data.vote is number

|| request.resource.data.randomNumber is number

&& request.resource.data.anonymous == resource.data.anonymous

&& request.resource.data.content== resource.data.content

&& request.resource.data.userName == resource.data.userName

&& request.resource.data.owner == resource.data.owner && request.auth.uid != null;

}
```

`vote` and `randomNumber` fields are updatable.

`anonymous` , `content` , `userName` , `owner` fields are not updatable.

Hopefully this article will solve your problems. Good work!

Firebase Firebaseauthentication The Cloud Security Rules Cloud Firestore Security