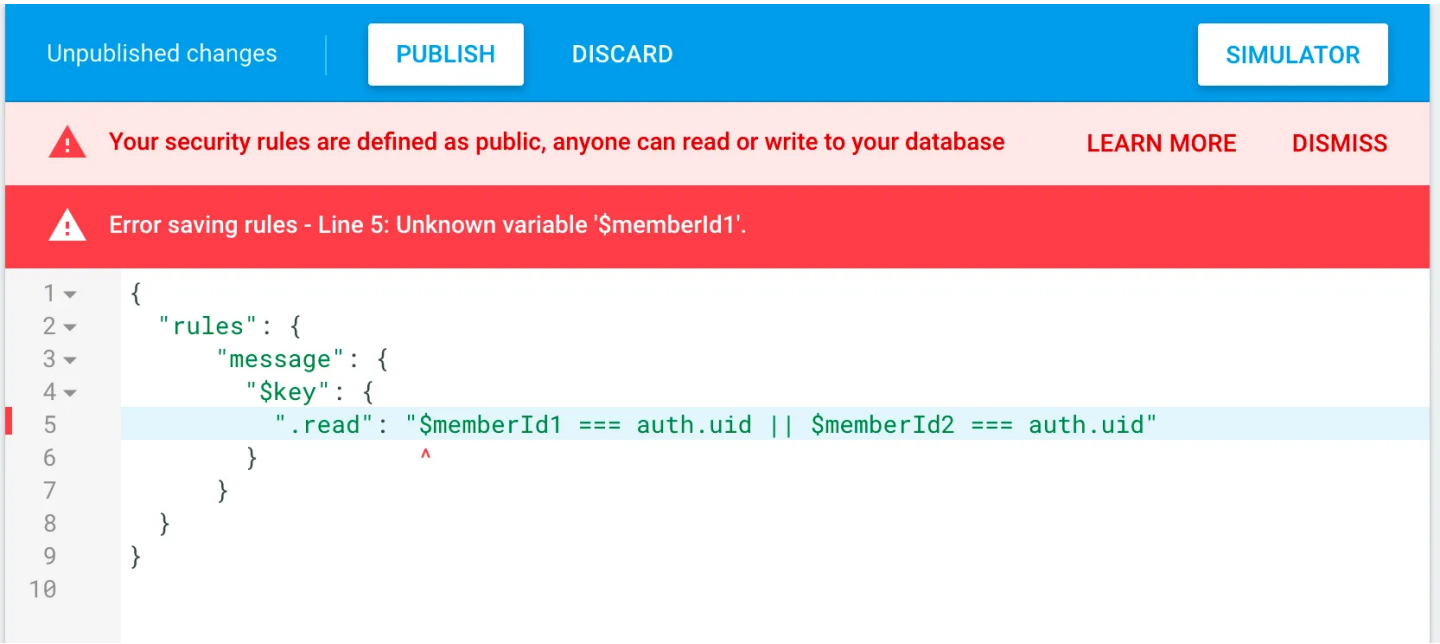
10 Firebase Realtime Database Rule Templates

4 min read · Jan 30, 2019

Julio Marín



Firebase Console Realtime Database Rules

Recently while playing with React+Firebase and the Firebase Realtime Database I had to read about its security and figured that Firebase Realtime Database provides a flexible language of rules based on expressions with a syntax similar to JavaScript that allows you to define how the data should be structured, how they should be indexed and when they can be read and written, all this in an easy way.

These rules are hosted on Firebase servers and are applied automatically at all times and you can change the rules of your database in Firebase console. You just have to select your project, click on the Database section on the left and select the Rules tab.

Rule Types

The rules have a JavaScript-like syntax that make it easy to understand and those

comes in four types:

.read
Describes if and when data is allowed to be read by users.

.write
Describes if and when data is allowed to be written.

.validate
Defines what a correctly formatted value will look like, whether it has child attributes, and the data type.

.indexOn
Specifies a child to index to support ordering and querying.

The Firebase Documentation Site is pretty good and if you want to get deep on this and really understand Firebase Realtime Database Rules, it is the place to go.

Find a list of common Firebase Realtime Database Rules you can use in your projects:

1) No Security

These rules give anyone, even people who are not users of your app, read and write access to your database.

During development, you can use the public rules in place of the default rules to set your files publicly readable and writable. This can be useful for prototyping, as you can get started without setting upAuthentication. **This level of access means anyone can read or write to your database. You should configure more secure rules before launching your app.**

```
// No Security

{
 “rules”: {
 “.read”: true,
 “.write”: true
 }
}
```

2) Full Security

These are the default rules that disable read and write access to your database by users. With these rules, you can only access the database through the Firebase console

```
// Full security

{
 “rules”: {
 “.read”: false,
 “.write”: false
 }
}
```

3) Only authenticated users can access/write data

```
// Only authenticated users can access/write data

{
 “rules”: {
 “.read”: “auth != null”,
 “.write”: “auth != null”
 }
}
```

4) User Authentication from a particular domain

```
// Only authenticated users from a particular domain (example.com)
can access/write data
```

```
{
 “rules”: {
 “.read”: “auth.token.email.endsWith(‘@example.com’)”,
 “.write”: “auth.token.email.endsWith(‘@example.com’)”
 }
}
```

5) User Data Only

Here’s an example of a rule that gives each authenticated user a personal node at `/post/$user_id` where `$user_id` is the ID of the user obtained through Authentication. This is a common scenario for any apps that have data private to a user.

```
// These rules grant access to a node matching the authenticated
// user's ID from the Firebase auth token
{
  "rules": {
    "users": {
      "$uid": {
        ".read": "$uid === auth.uid",
        ".write": "$uid === auth.uid"
      }
    }
  }
}
```

6) Validates user is moderator from different database location

```
// Validates user is moderator from different database location
{
 “rules”: {
 “posts”: {
 “$uid”: {
```

```
 ".write": "root.child('users').child('moderator').val() === true"
 }
 }
 }
}
```

7) Validates string datatype and length range

```
// Validates string datatype and length range
{
 "rules": {
 "posts": {
 "$uid": {
 ".validate": "newData.isString()
 && newData.val().length > 0
 && newData.val().length <= 140"
 }
 }
 }
}
```

8) Checks presence of child attributes

```
// Checks presence of child attributes
{
 "rules": {
 "posts": {
 "$uid": {
 ".validate": "newData.hasChildren(['username', 'timestamp'])"
 }
 }
 }
}
```

9) Validates timestamp

```
// Validates timestamp is not a future value
{
 “rules”: {
 “posts”: {
 “$uid”: {
 “timestamp”: {
 “.validate”: “newData.val() <= now”
 }
 }
 }
 }
}
```

10) Prevents Delete or Update

```
// Prevents Delete or Update
{
 “rules”: {
 “posts”: {
 “$uid”: {
 “.write”: “!data.exists()”
 }
 }
 }
}
```

BONUS: Prevents only Delete

```
// Prevents only Delete
{
 “rules”: {
 “posts”: {
 “$uid”: {
 “.write”: “newData.exists()”
 }
 }
 }
}
```

BONUS2: Prevents only Delete

```
// Prevents only Update
{
 “rules”: {
 “posts”: {
 “$uid”: {
 “.write”: “!data.exists() || !newData.exists()”
 }
 }
 }
}
```

BONUS3: Prevents Create and Delete

```
// Prevents Create and Delete
{
 “rules”: {
 “posts”: {
 “$uid”: {
 “.write”: “data.exists() && newData.exists()”
```

```
    }
    }
    }
  }
```

Hopefully this will be helpful for you and if you want to know more about the SalsaMobi education efforts on different technologies you can **subscribe here** and we will notified you when we create more articles or tutorials.