GunFire!
A Firebase persistence layer for GunDB

Oghenemaro Lemuel Okoro
14545163

Final Year Project – **2018**
B.Sc. Single Honours in Computer Science and Software Engineering

Department of Computer Science
Maynooth University
Maynooth, Co. Kildare
Ireland

A thesis submitted in partial fulfilment of the requirements for the B.Sc.
Single in Computer Science and Software Engineering.

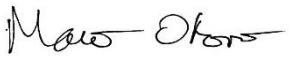
Supervisor: **Dr Kevin Casey**

Contents

Declarations

I hereby certify that this material, which I now submit for assessment on the program of study as part of the qualification, is *entirely* my own work and has not been taken from the work of others - save and to the extent that such work has been cited and acknowledged within the text of my work.

I hereby acknowledge and accept that this thesis may be distributed to future final year students, as an example of the standard expected of final year projects.

Signed:  Date: 28/03/18

Acknowledgements

Special thanks to the amazing friends that pushed me to never give up no matter how hard things got and never giving up on me. I am forever grateful and blessed for the help and love you have shown me throughout the years.

Abstract

GUN is a graph-based, NoSQL database, that allows you to sync and store data. The information is stored locally on the browser in a data.json file. Wherever there is JavaScript, GUN will run. GUN contains certain features such as being real-time (data that is saved in one machine will sync to other peers), distributed (peer-to-peer by design, no centralised database), offline first (GUN works while offline and when an internet connection is available, GUN synchronises the changes made offline) and a flexible database that uses graphs to let you have any type of data structure you want. Firebase is used for mobile and web development and contains a real-time database that allows for storing and syncing data on the firebase cloud. The data is stored as JSON and is synchronised across all clients. A persistence layer is a software layer that makes it easier for programs to persist their state. It serves as the connection to the hidden data source. For this project, Node.js (JavaScript on the server) is employed to build a GunDB adapter that acts as a persistence layer for GUN's data, thus, GUN's data is stored in the Firebase real-time database.

Chapter one: Introduction

1.1 The topic addressed in this project

The topic addressed in this project is to build a Firebase persistence layer for GunDB. This persistence layer will store Gun's data in the Firebase real-time database.

GUN allows for pluggable storage engines, to store its data and comes with backend Storage engines included such as: [1]

Included Storage Engines [1]

- File: the default way to store GUN's data but is used for development purposes
- Radix: recently introduced to GUN and will soon be the default way of storing
- Amazon S3: (broken) Uses Amazon S3 to store data

Third Party Storage Solutions [1]

- Level: Uses LevelDB as a storage adapter to store GUN's data (done with the interface LevelUP)
- File System (Alt1): A file streaming adapter for GUN
- SQLite/ODBC (SACK): The default storage engine is set to an SQLite DB file.
- Flint: Allows integrations with other databases as a backend storage engine for GUN. For example, MongoDB, MySQL, Cassandra, Elasticsearch

1.2 Motivation

Though there is a list of backend storage engines listed above, it is ideal to use Firebase because it has similar features to GUN such as being a NoSQL Database and storing the data in graphs (JSON[1]). Firebase has well laid out documentation as well as a console, to view the real-time database data.

The motivation for this project is to find a way to plug firebase as a backend storage engine for GUN's data, this way GUN's data is synced, stored and viewed from the Firebase console. Users can then utilise GUN in their applications while Firebase handles storage. Available to them is a free Firebase GunDB adapter that can persist Gun's data to Firebase' scalable real-time database. Therefore, getting the best of both GUN and Firebase.

A Firebase persistence layer for GUN reduces the possibility of vendor lock-in[2]. If someone is using Firebase, they could move their code slowly to GunDB bit by bit since GunDB and Firebase can share the same data.

---

[1] *"syntax for storing and exchanging data"* [2]

[2] Makes a customer dependent on a certain product for services and they're unable to switch to a different product without making switching costs.

1.3 Problem statement

The problem is there is no way to plug firebase as a backend storage engine with GUN. There are also no examples of GUN and Firebase working together. GUN is seen as Firebase but with added functionality and the developers of GUN probably hadn't thought of any sort of way for GUN and Firebase to communicate.

1.4 Approach

The solution to this problem is to build a Firebase storage adapter that allows Firebase to persist GUN's database. To do this the MongoDB adapter for GUN (gun-mongo) will be used as a template to assist in building the Firebase adapter for GUN. This will aid in understanding how the code works. When coding the functions for the Firebase adapter the parameters, *"opt*", "*get"* and "*put",* are coded in a way that Firebase can connect, read and write to the desired database.

The GUN module, gun-flint, is utilised in building the adapter. This node module aids in making the process of building the adapter easier by making sure GUN's data is in a form that the adapter can recognise when they communicate.

1.5 Metrics

The adapter is tested by doing unit tests to evaluate how the adapter connects, reads and writes to the Firebase database. With the adapter plugged into GUN the "`.get ()`" and "`.put ()`" are tested to see if they can handle complex operations. Due time constraints the performance of the adapter is not tested against other available gun adapters to see if the gun-firebase adapter offers better performance, but it something that can be done in the future.

1.6 Project

In the process of creating the adapter the achievements made are:

- There is now a way for users to store their GUN data on Firebases' real-time database via a gun-firebase adapter.

- There is a successful connection with the gun-firebase adapter and the Firebase real-time database.

- The adapter is able to read and write data to the Firebase real-time database albeit under specific circumstances.

Chapter two: Technical Background

2.1 Technical material

In this section, the technical materials are discussed. The materials listed below are utilised in creating the adapter

Node.js: A JavaScript run-time environment used to execute server-side JavaScript code. It is memory efficient as it uses asynchronous programming. It eliminates any waiting and continues with requests as they come. It can generate page content that is dynamic and present it to web clients, can read, write, update and delete files, on the server. It collects data as forms from the client side and can connect to various databases and perform operations on the databases.

Node.js uses modules to make it easier to create complex applications. Modules are like libraries. They contain a set of functions that are related to the subject of that module. Calling the "`require ()`" function returns a reference to the module entered e.g. `var http = require ('http')` will search for the http module in the node_modules folder and now http is referenced in the Node.js code. [4]

NPM: Node package manager is for Node.js modules. On npmjs.com there are plenty of Node.js packages called the npm registry. A package in Node.js contains all the files needed for a module and the modules can then be included in a project. [5]

GUN: GUN is a graph-based storage system for JavaScript. It stores data locally on the browser in a data.json file and has a pluggable interface which can be used to switch GUN's storage. For example, it could store on the Amazon S3 Storage. It has a peer-to-peer distributed architecture meaning that PC's connected to the GUN server share resources without going through a separate server computer. GUN solves state synchronisation (sending both input and state) using their conflict resolution algorithm called HAM, thus, GUN synchronises regardless of updates to the data. [6]

GUN-level: A LevelDB backend storage adapter for GUN. It has an interface called LevelUP that has all the methods of a database such as get and put. Level has a vast ecosystem and there are a lot of plugins, backends, and utilities made for level. [7]

GUN-mongo: A MongoDB adapter for GUN. Each graph node is stored in a document and the key of the node is '_id'. It is good for small to medium nodes and used more for nodes being created rather than updated. [8]

GUN-mongo-key: A key: value MongoDB adapter for GUN. Unlike gun-mongo, gun-mongo-key stores each graph node's key: value. This is good for updates and streaming large amounts of nodes. [9]

GUN-flint: A package that makes it easy to build adapters to connect GUN's database to other services but is not an adapter itself. It does this by making sure the data from Gun is in a consistent format for the adapter and that the information the adapter returns is in a format GUN perceives. [10]

Robo3T/Robomongo: A visual tool that helps with managing the database, MongoDB. It is a free open source software. It embeds the actual mongo shell in a tabbed interface with access to a shell command line as well as GUI interaction. [16]

Firebase: A powerful web and mobile and application platform owned by Google. It has great tools for developing web and mobile apps such as; the real-time database (stores data real-time in a JSON tree format), authentication (Used for authenticating users into an app), cloud storage (used to store client content such as images, audio and videos) and hosting (used to host the app using the command line and can also create a custom domain for the app). [11]

The programs listed below are programs that didn't work (Fired-up and Amazon S3 Storage) and programs that were thought to be a solution to minor errors (Browserify and RequireJS).

Fired-up: A Firebase plugin based on LevelDB/LevelUP. [3]

Amazon S3 Storage: A web service offered by Amazon that has a scalable storage system and comes bundled with GUN. [12]

Browserify: JavaScript tool that allows developers write Node.js scripts that can compile to be used in the browser. It lets use the require function in the browser.

RequireJS: A JavaScript file and module loader. It is for in-browser usage. Installing allows for the "`require ()`" function to be called in the browser.

Chapter three: The Problem

3.1 Problem analysis

The main problem is that there are no examples or resources that show GUN and Firebase working together in anyway but rather how they compare form each other and which one would be better for individuals to use in their project. GUN is an open source community GitHub project and so resources on GUN's website are still being changed, modified and moved. They are constantly making changes to the code.

GUN does provide a template for creating adapters but there's still a matter of finding what operations will specifically work with Firebase. The way in which the Firebase adapter will connect, read and write to the Firebase real-time database, is different from the way other adapters do.

Oher problems that come up are:

- Understanding how both GUN and Firebase can be utilised in an app, how they are installed, how to connect to their databases, how to read and write data with them and ultimately how to make them work in sync.

- Though there are already adapters made for GUN such as gun-mongo and gun-level. They can only really be used as a template for building a Firebase adapter. They are both very different code and the author of both codes is the author of GUN. The level adapter is an adapter built from scratch while the mongo adapter is built with gun-flint.

```
/* eslint-disable id-length*/
import Gun from 'gun/gun';
const writing = Symbol('In-process writes');
const notFound = /(NotFound|not found|not find)/i;
const options = {
  valueEncoding: 'json',
};

// Gun merge algorithm, authored by Mark Nadal.
const union = (vertex, node, opt) => {
  if (!node || !node._) {
    return;
  }
  vertex = vertex || Gun.state.to(node);
  if (!vertex || !vertex._) {
    return;
  }
  opt = Gun.num.is(opt) ? { machine: opt } : { machine: Gun.state() };
  opt.union = Gun.obj.copy(vertex);
  if (
    !Gun.node.is(node, function(val, key) {
      const HAM = Gun.HAM(
        opt.machine,
        Gun.state.is(node, key),
        Gun.state.is(vertex, key, true),
        val,
        vertex[key],
      );
      if (!HAM.incoming) {
        return;
      }
      Gun.state.to(node, key, opt.union);
    })
  ) {
    return;
  }

  return opt.union; // eslint-disable-line
};
```

Figure Gun-level code on the gun-level GitHub page. [7]

```
const {NodeAdapter} = require('gun-flint');
const Mongojs = require('mongojs');

module.exports = new NodeAdapter({

    /**
     * @type {boolean}  Whether or not the adapter has been properly initialized and can attempt DB connections
     */
    initialized: false,

    /**
     * Handle Initialization options passed during Gun initialization of <code>opt</code> calls.
     *
     * Prepare the adapter to create a connection to the Mongo server
     *
     * @param {object}  context    The full Gun context during initialization/opt call
     * @param {object}  opt        Options pulled from fully context
     *
     * @return {void}
     */
    opt: function(context, opt) {
        let mongo = opt.mongo || null;
        if (mongo) {
            this.initialized = true;
            let database = mongo.database || 'gun';
            let port = mongo.port || '27017';
            let host = mongo.host || 'localhost';
            let query = mongo.query ? '?' + mongo.query : '';
            this.collection = mongo.collection || 'gun-mongo';
            this.db = Mongojs(`mongodb://${host}:${port}/${database}${query}`);

            this.indexInBackground = mongo.indexInBackground || false;
        } else {
            this.initialized = false
        }
    },
```

Figure Gun-mongo code on the gun-mongo GitHub page. [8]

- In the future, the libraries and dependencies of Firebase could change meaning that the code of GUN and the gun-firebase adapter may not be compatible. The adapter will need to keep up with these updates that are made to GUN and Firebase or it could easily become obsolete

- Trying to get the bundled storage (Amazon S3) to work with GUN is not possible as there are errors in the GUN code and S3 is currently broken. The screenshot shown below is in the file s3.js in the GUN node folder. GUN developers are currently trying to move away from using this method of storing.

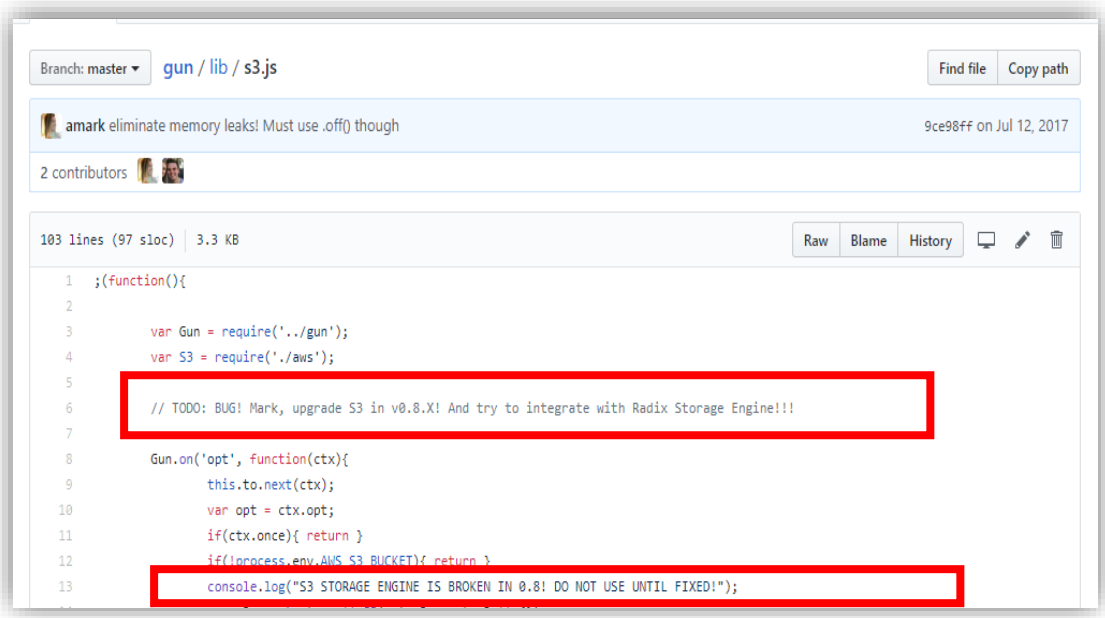


Figure Amazon S3 code on GUN's GitHub page

- There is a package on npmjs.com called fired up ("*A node.js implementation of firebase based on LevelDB/LevelUP*" [3]). Installing the module proves difficult and on emailing the author of the code for suggestions on how to fix the errors, the author replied saying "*the project was a bit more of a proof of concept I built many years ago, it probably doesn't work very well as the libraries and dependencies are quite out of date".*

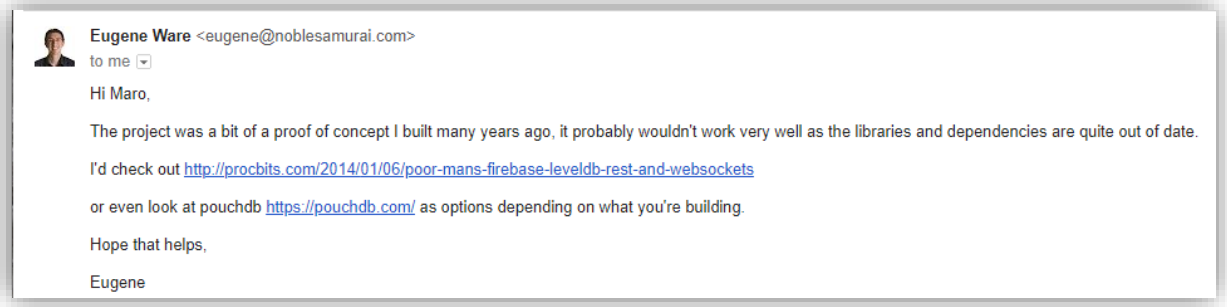


Fig Email of fired up npm module explaining why the package does not work

Chapter four: The Solution

The solution to this problem is to build a Firebase storage adapter that allows Firebase to persist GUN's database. There are many ways to build an adapter but, using the package "GUN-Flint", is an easy way to connect the adapter to GUN's database. GUN, being a graph database, needs flexibility when the adapter is storing information. GUN-Flint provides this flexibility. This allows the adapter to store data in a way that the chosen storage system can understand.

When building the adapter, the three methods needed are:

- "`Opt`" – This method is used to connect the adapter to the server and make database connections. It takes in two parameters; context (The full GUN context during initialization/opt call) and opt (Options pulled from full context).

- "`Get`" – This method reads/retrieves what's in the database. It has two parameters; key (The key for the node to retrieve) and done (Call after retrieval or error).

- "`Put`" – This method writes nodes to the databases. It has three parameters; key (The key for the node), node (The full node with metadata), done (Called when the node has been written).

There are also 3 methods of storing data with a GUN-Flint adapter:

- Node Storage – The get method reads an entire GUN node while the put method writes an entire node.

- Key: Value - The get method returns an array/list of nodes properties; put writes a batch of updates to specific node's Key: Value pairs.

- Delta Storage: The get method returns an entire node that is formatted comprehensively for GUN; the put method receives a delta (a diff[3] that is produced at the time of the put request) of node properties as well as conflict-resolution state indicators.

Each storage method has their pros and cons for using them, for example, Node Storage and Key: Value are easy to implement and there are no additional concerns for conflict resolution but Key: Value is faster than Node Storage as it is not storing just one node each time but a batch of nodes. Delta Storage is the most flexible of the three storage methods and gives the best performance but it is difficult to implement and if done incorrectly it can lead to data corruption during conflict resolution.

The approach is to use the MongoDB adapter as a template for making the Firebase adapter. First Gun is installed with the npm command: *"npm install gun".* The name of the MongoDB adapter is gun-mongo and is installed with the npm command: "*npm install gun-mongo"*. Gun-mongo uses node storage to store GUN's data in a document. First, it is necessary to check that the gun-mongo adapter is working.

[3] Diff – delta encoding where the data is stored in the form of differences between sequential data rather than complete files. It calculates and displays the differences between two files.

4.1 Gun-mongo adapter set up[4]

The Gun-mongo.js code is opened in a text editor to understand how the code works. A server.js file is created to test out the mongo adapter like so: (shown in the screenshot in the appendices). Variables are added to server.js to test the gun-mongo adapter. Robomongo is installed with the help of this video: [13]

To make a connection to MongoDB enter the command: *"sudo service mongod start"* on the console. The connection is verified by searching the contents of the log file "/var/log/mongodb/mongod.log" for a line reading: *[initandlisten] waiting for connections on port <port>* (<port> is the port configured in /etc/mongod.conf, 27017 by default) [15]. To stop the connection the command: "*sudo service mongod stop*". Robomongo is opened to make sure there is a connection to the port.

In the console, the command: "*node server*" is entered. In Robomongo it shows the key 'bob' and 'dave' have been entered as well as their values. Now, it is confirmed gun-mongo works.

4.2 The Gun-firebase adapter construction [5]

Now that the gun-mongo adapter is confirmed to be working, the gun-firebase adapter can be constructed.

The gun-mongo adapter code will be used as scaffolding for the gun-firebase adapter. Though the way in which the gun-mongo adapter works is different to how the gun-firebase adapter will work the gun-mongo adapter code is a good template in understanding how certain parameters in the code work such as "`key` and "`done`". There are comments added to the gun-mongo code already so it easy to understand how the code works and this in turn aids in making the process of building the gun-firebase adapter easier.

Gun-flint is installed with the npm command: "*npm install gun-flint*". A folder called "gun-firebase" created in the node_modules. Using a text editor, a document called gun-firebase.js and the code (shown in the appendix). Gun-flint and Firebase are "`required`" in the JavaScript document. The main functions in the code are "`opt`", "`get`" and "`put`".

The "`opt`" function is used to make a connection to the Firebase real-time database. It utilises the Firebase apikey, authorisation domain and databaseURL to connect to the Firebase real-time database. If the connection is successful there is a console message "connected to the database".

The "`get`" function gets a reference to the database and uses the reference to get a "`snapshot`" of the specific key in the real-time database. The done function is calling null meaning that when get is done it will now call the put function. There is also an error call-back included in the get code in case the reader does not work.

[4] The screenshots for this section are in the appendix
[5] The screenshots for this section are in the appendix

The "`put`" function gets a reference to the database and sets the value in Firebase' real-time database. The "`.set ()`" Firebase function sets the value of the key to the object (node). Node has symbols that firebase can't read so delete node removes the symbols so Firebase can set the value. If the write is successful the console will print the new value and send a confirmation message.

4.3 Gun-firebase adapter in action[6]

In the server.js file, Firebase is initialised with the `apiKey, authDomain, databaseURL` of the project that has been created on Firebase. Local Storage is set to false to stop Gun from storing its data locally in a data.json file. On the console, the command: "*`node serve`*" is entered and the key `'bob'` and `'dave'` are stored in firebase as well as their values. The console prints out the actions going on (shown in the screenshot in the appendix).

[6] The screenshots of this section are in the appendix

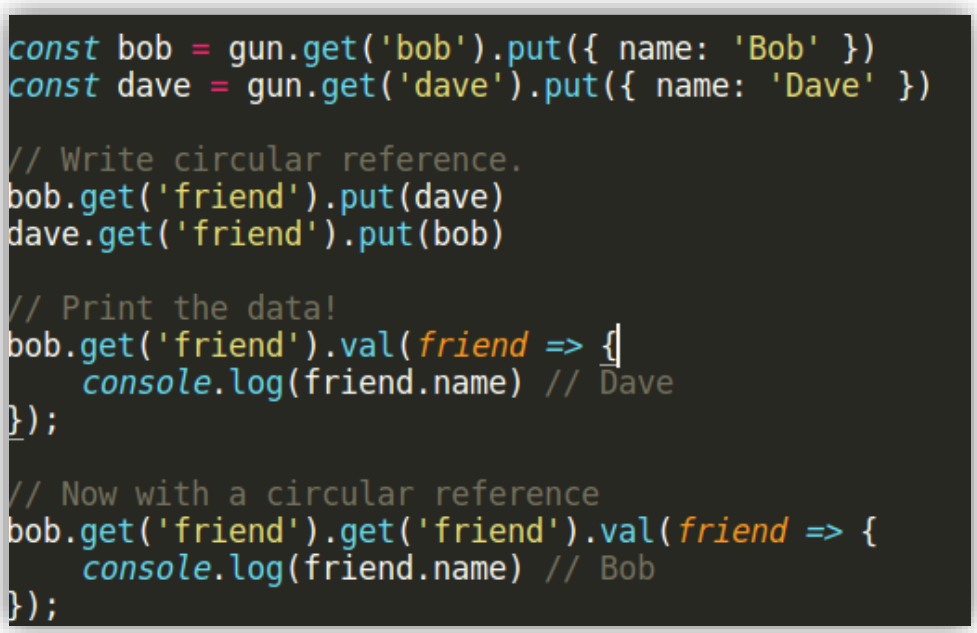
Chapter five: Evaluation

5.1 Software Verification

The test approach is unit testing, to see how each part of the gun-firebase code works. The "*opt*" function, is tested for connection to the database. The "*get"* function, is tested for reading the database. The "*put"* function, is tested for writing to the database. This code snippet is used to test the adapter. To get the adapter start the command: `node server.js` is typed into the console. The code for connecting to the specific database.

```
var Gun = require('gun');
var firebase = require('gun-firebase');

var gun = new Gun({
localStorage: false,
 gunfirebase: {
    apiKey: "AIzaSy
    authDomain: "th
    databaseURL: "h
    projectId: "th
    storageBucket:
    messagingSende
    }

});

const bob = gun.get('bob').put({ name: 'Bob' })
const dave = gun.get('dave').put({ name: 'Dave' })
```

Server.js code



```
const bob = gun.get('bob').put({ name: 'Bob' })
const dave = gun.get('dave').put({ name: 'Dave' })

// Write circular reference.
bob.get('friend').put(dave)
dave.get('friend').put(bob)

// Print the data!
bob.get('friend').val(friend => {
    console.log(friend.name) // Dave
});

// Now with a circular reference
bob.get('friend').get('friend').val(friend => {
    console.log(friend.name) // Bob
});
```

Screenshot of a circular reference being done in the Server.js file

Test cases	Input	Expected Output	Actual Output	Test Outcome
Require "gun" and the adapter, "gun-firebase". Enter the apikey, authDomain, and DatabaseURL in the gun constructor as shown above	`apikey, authDomain, and DatabaseURL`	A Console message saying there is a connection to the database is printed	The adapter is connecting to the Firebase database	Pass, Connection to the database
Enter the apikey, authDomain, and DatabaseURL in the gun constructor as shown above. Don't require "gun" and "gun-firebase"	`apikey, authDomain, and DatabaseURL`	Reference Error, gun is not defined	There is no gun reference, therefore, gun functions can't be used and no connection to the database can be made	Fail, no connection to the database is made as the gun module and adapter aren't being referenced

Enter the projectId, storageBucket and messagingSenderId of the Firebase Project	`projectId, storageBucket and messagingSenderId`	The database still connects successfully	There is a connection to the database because the adapter only needs the `apikey, authDomain,` and `DatabaseURL`	Pass, A connection to the database is still made
Enter all parts of the code as shown in the screenshot above	The key `bob` and the object "*`name: 'Bob'`*". The key `dave` and the object "`name: 'Dave'`".	The keys bob and dave is entered into the Firebase real-time database as well as their objects	The `get` and `put` methods work for this operation	Pass, the keys bob and dave are in the Firebase real-time database as well as their objects
Take out the "put" function in the constant bob and dave	The keys "`dave`" and "`bob`"	The values stored in the keys "`dave`" and "`bob`" are displayed in the console	The get function needs another function to accompany it such as put or else it will not work	Fail, the Firebase adapter calls get but nothing is retrieved
Type the circular reference code shown in the screenshot above	The keys "`dave`" and "`bob`"	The key bob now stores the value object "`name: Dave`" whilst the key dave now stores the value object "`name: Bob`"	Firebase is issuing an error to the console	Fail, the keys values/objects are not swapped.

From testing, the adapter works in the specific case where "`.get ()`" and ".put ()" are used in a simple way. When trying to reference circularly or do any more complex operations, there are multiple errors that are displayed in the console.

The main issue causing errors is that GUN is still storing data locally. The line that turns off the local storage (`localStorage: false)` is not working as it is supposed to. The data.json file is still being created and is storing GUN's nodes there as well as on Firebase. When doing a "`.get ()`" followed by "`.on ()`" or even a ".`val ()`", the data is conflicted. Get is communicating with Firebase and the data.json file and "`.on ()`" and ".`val ()`" are communicating with the data.json file only.

Chapter six: The Conclusion

6.1 Contribution to the state-of-the-art

In creating this gun-firebase adapter it adds to the collection of adapters available in GUN's inventory. People now have a way they can persist their GunDB data to free Real-time scalable database (Firebase).

6.2 Results discussion

The results achieved in the adapter are specific to a particular case i.e. where get and put are the only functions called in the server.js file. When trying to use another gun function such as "`.on ()`" or "`.val ()`" the console gives all sorts of errors. This is because the line "*`localStorage: false`*" is supposed to turn off local storage. Unfortunately, this line is not working as each time the server.js code is run a data.json file is being created in the directory being worked in. Also, when trying to do a circular reference, there are errors in the console[7]. When trying to circularly reference the keys 'dave' and 'bob' and print them in the console; gun can do it locally, but, it cannot do it with GUN. This is because there are conflicts with GUN storing locally and Firebase storing.

Firebase also has limitations to its real-time database. To list some[8]: [14]

- The size of a single event triggered by a write is 1MB. Writes bigger than 1MB can write successfully but they don't trigger a function invocation.
- The limit to the length of a key is 768 bytes, and can't contain characters such as (. $ # [] /) or any ASCII control characters.
- The length of time a single query can run is 15 minutes before failing.
- The limit of the size of a single write request to the database is 16MB from the SDKs.
- The limit of the size of database downloaded from the database at a single location should be less than 256MB for each read operation.

[7] Screenshot is in the appendix
[8] Screenshots of the Firebase real-time database limitations are in the appendix

6.3 Project approach

The approach of using the gun-mongo adapter as scaffolding for the gun-firebase adapter is somewhat resourceful. The way in which gun-mongo reads and writes its data is different to the way Firebase would read and write its data. Gun-mongo is storing each graph node in a document whilst the Firebase real-time database is a NoSQL cloud database.

The gun-mongo adapter does aid in understanding how the gun-mongo adapter works which does help in understanding how it is reading and writing data. This understanding is used in building the gun-firebase adapter.

Using gun-flint also aids in not having to build the adapter completely from a scratch, which takes a lot longer to do and is confusing to understand, and helps with GUN and the adapter communicating.

6.4 Future work

Based on the work done on the gun-firebase adapter, there is still a lot of work to be done before the adapter could be used for more than development

There's still the matter of doing performance tests to check the speed of the writes and reads to the database. Being a node storage type of storage adapter, it is the slowest methods of storing data with a gun-flint based adapter. If the data were to require large nodes, it could overwhelm the memory and crashes will occur. A better solution would be to make the adapter with the Delta Storage method. This way, there is more control of the storage format, there is a much higher performance possibility and there is more flexibility with how data is stored. The only caveat to using this method is that it is difficult to implement.

There are also special GUN testing suites that can be utilised to further test the adapter. Flint is packaged with its own CLI tool that runs tests on adapters. Unfortunately, due to time constraints, these tests were not carried out. The adapter has also not been used in applications yet. It is unclear how the adapter will act when used in applications.

Further improvements will also need to be made in the future such as maybe re-working the gun-firebase get and put methods. To read and write data more effectively. In the future the GUN code could change so the adapter interface works well, the documentation could be more descriptive to make it easier for anyone to build an adapter. The adapter will also need to be updated in case there were significant changes made to GUN or gun-flint.

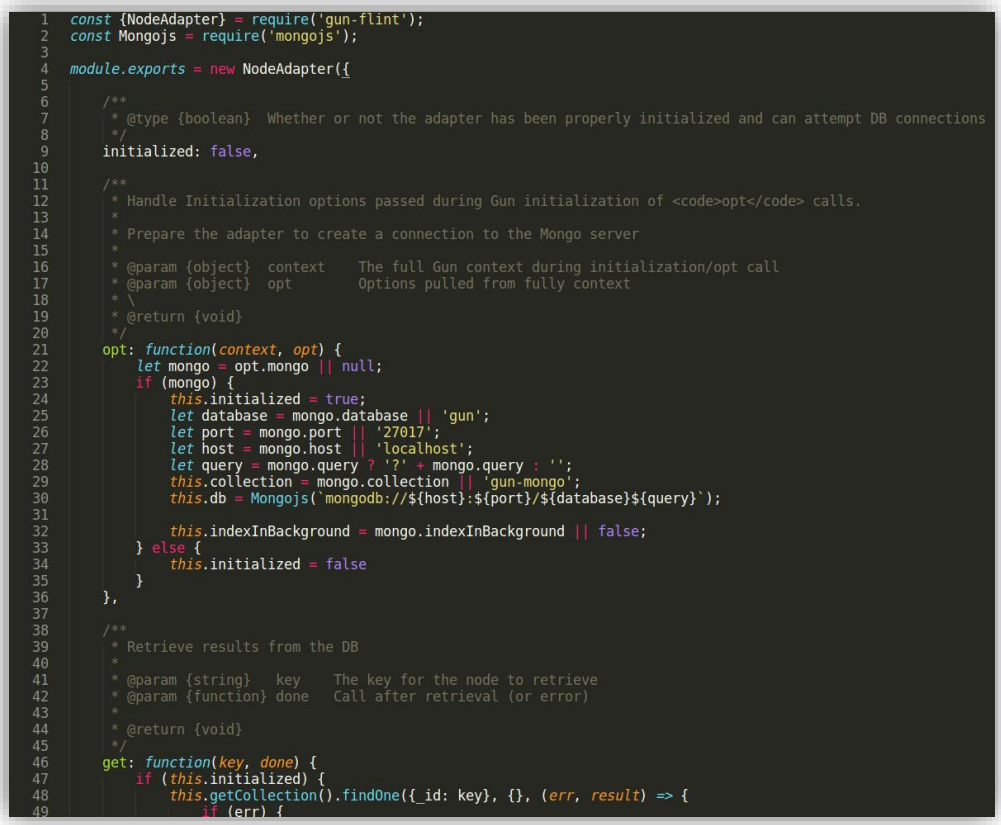
An alternative approach in the future would be to utilise the Firebase storage bucket to store the data.json file that GUN was using to store its nodes. The file could be in the storage bucket and any changes made to GUN's data would be registered to the data.json file (now stored in firebases storage bucket). Firebase is currently building a new type of storage called "Cloud Firestore". It is like the real-time database only this database will have more powerful queries. It stores data in documents and organises the data into collections (like MongoDB).

References

1. W3schools: *JSON – Introduction*, https://www.w3schools.com/js/js_json_intro.asp: accessed September 2017
2. GUN: *Storage: included storage engines and third-party storage solutions*, https://GUN.eco/docs/Storage: accessed September 2017
3. Firedup module: *A node.js implementation of firebase based on LevelDB/LevelUP*, https://www.npmjs.com/package/firedup: accessed October 2017
4. List of links for Node.js:
   Envato tuts+, https://code.tutsplus.com/tutorials/nodejs-for-beginners--net-26314: accessed October 2017
   TutorialKart, https://www.tutorialkart.com/nodejs/nodejs-tutorials/: accessed by October 2017
   W3schools Node.js Tutorials, https://www.w3schools.com/nodejs/nodejs_intro.asp: accessed by October 2017
   W3schools Node.js Tutorials (built-in modules), https://www.w3schools.com/nodejs/ref_modules.asp: accessed by October 2017
5. Node package manager: *Library of Node.js packages*, https://www.npm.js.com: accessed by September 2017
6. GUN: *Intro website*, https://gun.eco: accessed September 2017
7. GUN-level: *LevelDB adapter for GUN*, https://www.npmjs.com/package/gun-level: accessed by September 2017
8. GUN-mongo: *MongoDB adapter for GUN*, https://www.npmjs.com/package/gun-mongo: accessed by September 2017
9. GUN-mongo-key: *A key: value MongoDB adapter for GUN*, https://www.npmjs.com/package/gun-mongo: accessed by September 2017
10. GUN-flint: *for writing your own Gun adapter*, https://www.npmjs.com/package/gun-flint: accessed by October 2017
11. Firebase: *helps you build better mobile and web apps*, https://firebase.google.com: accessed by September 2017
12. Amazon S3 Storage: *Amazon storage platform*, https://aws.amazon.com/s3: accessed by October 2017
13. How to install Node.js, MongoDB and Robomongo in Ubuntu 16.04 LTS, https://www.youtube.com/watch?v=AgsWWdhuKoQ: accessed by October 2017
14. Firebase documentation: *Real-time Database Limitations*, https://firebase.google.com/docs/database/usage/limits: accessed by February 2018
15. MongoDB documentation, https://docs.mongodb.com: accessed by October 2017
16. Robomongo, https://robomongo.org/: accessed by November 2017

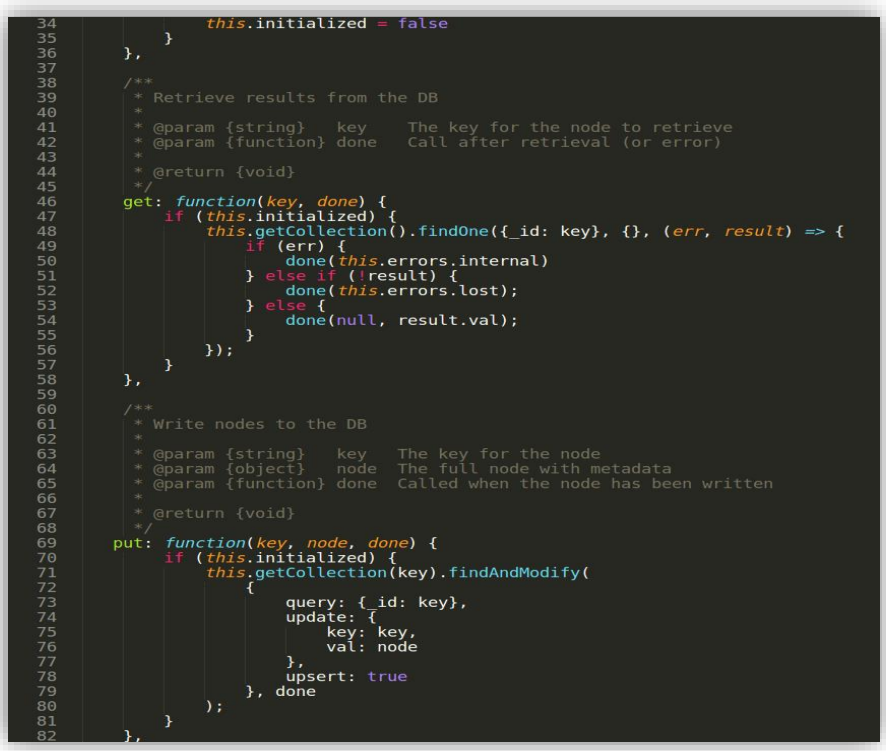
Appendices

Appendix1 Gun-mongo adapter set up screenshots



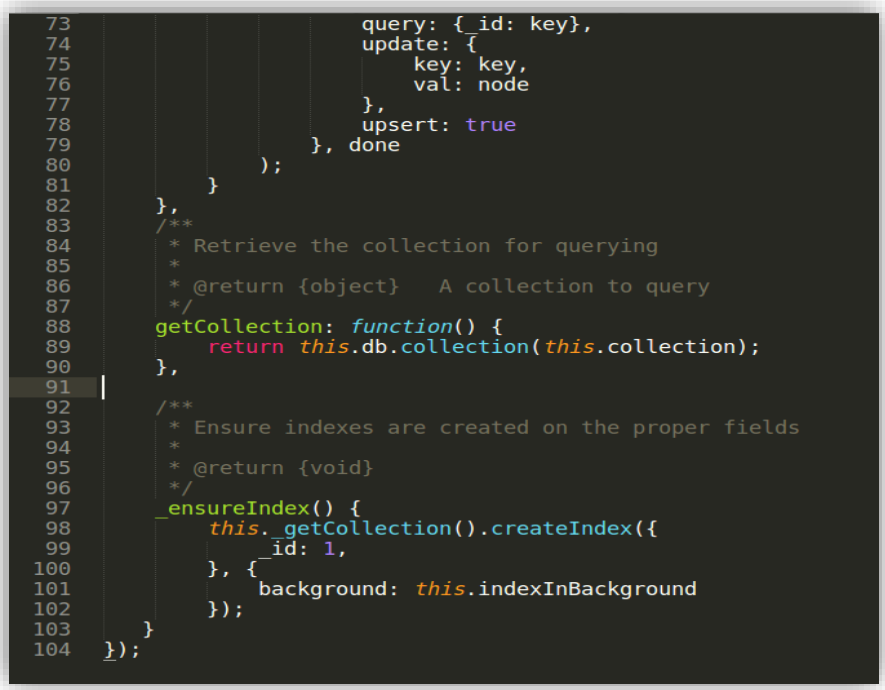
```
const {NodeAdapter} = require('gun-flint');
const Mongojs = require('mongojs');

module.exports = new NodeAdapter({

    /**
     * @type {boolean}  Whether or not the adapter has been properly initialized and can attempt DB connections
     */
    initialized: false,

    /**
     * Handle Initialization options passed during Gun initialization of <code>opt</code> calls.
     *
     * Prepare the adapter to create a connection to the Mongo server
     *
     * @param {object}  context    The full Gun context during initialization/opt call
     * @param {object}  opt        Options pulled from fully context
     * \
     * @return {void}
     */
    opt: function(context, opt) {
        let mongo = opt.mongo || null;
        if (mongo) {
            this.initialized = true;
            let database = mongo.database || 'gun';
            let port = mongo.port || '27017';
            let host = mongo.host || 'localhost';
            let query = mongo.query ? '?' + mongo.query : '';
            this.collection = mongo.collection || 'gun-mongo';
            this.db = Mongojs(`mongodb://${host}:${port}/${database}${query}`);

            this.indexInBackground = mongo.indexInBackground || false;
        } else {
            this.initialized = false
        }
    },

    /**
     * Retrieve results from the DB
     *
     * @param {string}   key    The key for the node to retrieve
     * @param {function} done   Call after retrieval (or error)
     *
     * @return {void}
     */
    get: function(key, done) {
        if (this.initialized) {
            this.getCollection().findOne({_id: key}, {}, (err, result) => {
                if (err) {
```

Screenshot of the Gun-mongo.js adapter code part 1



```
            this.initialized = false
        }
    },

    /**
     * Retrieve results from the DB
     *
     * @param {string}   key    The key for the node to retrieve
     * @param {function} done   Call after retrieval (or error)
     *
     * @return {void}
     */
    get: function(key, done) {
        if (this.initialized) {
            this.getCollection().findOne({_id: key}, {}, (err, result) => {
                if (err) {
                    done(this.errors.internal)
                } else if (!result) {
                    done(this.errors.lost);
                } else {
                    done(null, result.val);
                }
            });
        }
    },

    /**
     * Write nodes to the DB
     *
     * @param {string}   key   The key for the node
     * @param {object}   node  The full node with metadata
     * @param {function} done  Called when the node has been written
     *
     * @return {void}
     */
    put: function(key, node, done) {
        if (this.initialized) {
            this.getCollection(key).findAndModify(
                {
                    query: {_id: key},
                    update: {
                        key: key,
                        val: node
                    },
                    upsert: true
                }, done
            );
        }
    },
```

Screenshot of the Gun-mongo.js adapter code part 2

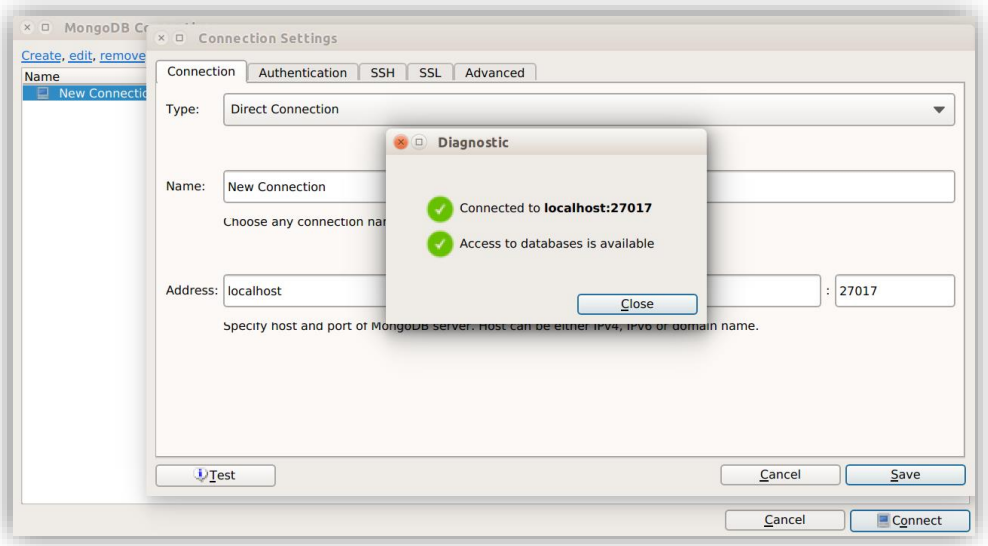


```
                    query: {_id: key},
                    update: {
                        key: key,
                        val: node
                    },
                    upsert: true
                }, done
            );
        }
    },
    /**
     * Retrieve the collection for querying
     *
     * @return {object}   A collection to query
     */
    getCollection: function() {
        return this.db.collection(this.collection);
    },

    /**
     * Ensure indexes are created on the proper fields
     *
     * @return {void}
     */
    _ensureIndex() {
        this._getCollection().createIndex({
            _id: 1,
        }, {
            background: this.indexInBackground
        });
    }
});
```

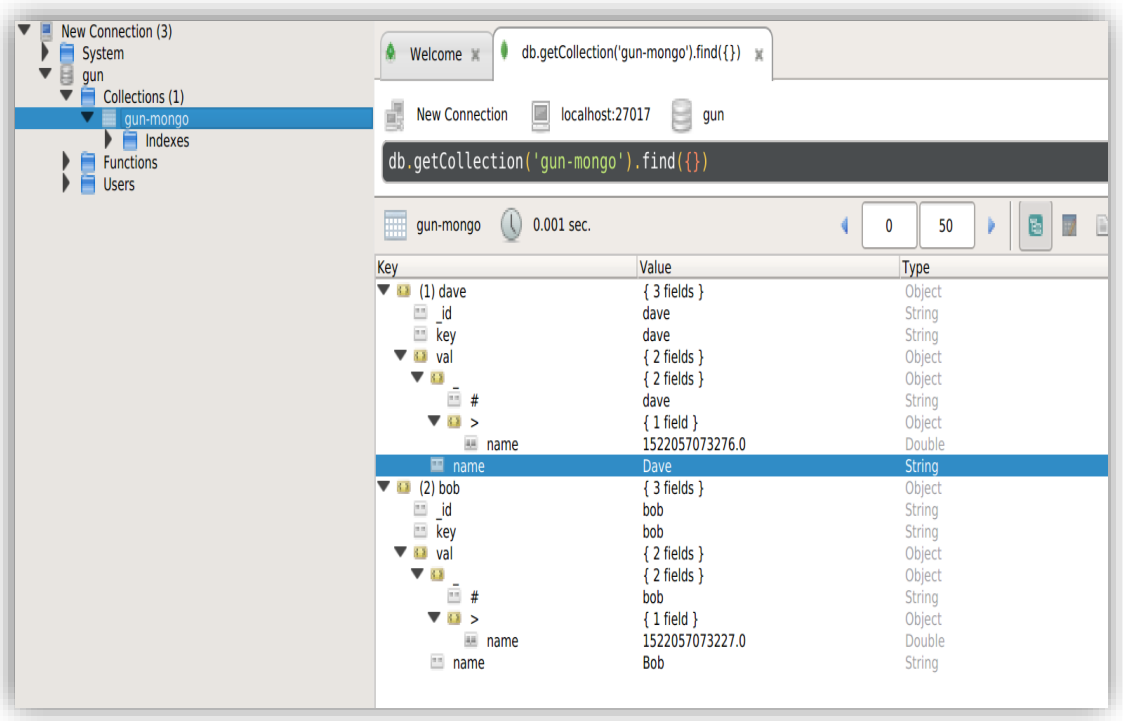
Screenshot of the Gun-mongo.js adapter code part 3

```
var Gun = require('gun');
//must be added after Gun but before instantiating Gun
var gunmongo = require('gun-mongo')
//Instantiate Gun
var gun = new Gun({
    file: false, // makes sure gun is not storing locally on data.json
    //these are default values
     mongo: {
        host: 'localhost',
        port: '27017',
        database: 'gun',
        collection: 'gun-mongo',
        query: ''
    }

});
```

Server.js file



Robomongo is connected to localhost: 27017 and there is access to databases



At the collection gun-mongo the key ‘bob’ and ‘dave’ are stored along with their values

Appendix2 Gun-firebase adapter construction screenshots

```
const {Flint, NodeAdapter} = require('gun-flint');
const firebase = require('firebase');

const GunFirebase = new NodeAdapter({

    /**
     * @type {boolean}  Whether or not the adapter has been properly initialized and can attempt DB connections
     */
    initialized: false,

    /**
     * Handle Initialization options passed during Gun initialization of <code>opt</code> calls.
     *
     * Prepare the adapter to create a connection to the Firebase server
     *
     * @param {object}  context    The full Gun context during initialization/opt call
     * @param {object}  opt        Options pulled from fully context
     *
     * @return {void}
     */
    opt: function(context, opt) {
        console.log("initialized", opt)

        let gunfirebase = opt.gunfirebase || null;
        firebase.initializeApp(gunfirebase);
        if (firebase) {
            this.initialized = true;
            let apiKey = firebase.apiKey;
            let authDomain = firebase.authDomain;
            let databaseURL = firebase.databaseURL;
            this.db = firebase.database();
            console.log("connected to the database")


        } else {
            this.initialized = false
        }
    },
    /**
     * Retrieve results from the DB
     *
     * @param {string}   key    The key for the node to retrieve
     * @param {function} done   Call after retrieval (or error)
     *
     * @return {void}
     */

   get: function(key, done) {
        console.log("calling get")
```

Screenshot of the gun-firebase adapter part 1

```
    } else {
        this.initialized = false
    }
},
/**
 * Retrieve results from the DB
 *
 * @param {string}   key    The key for the node to retrieve
 * @param {function} done   Call after retrieval (or error)
 *
 * @return {void}
 */

get: function(key, done) {
    console.log("calling get")

    //console.log(done.toString())
    if (this.initialized) {
        var db = firebase.database();
       var ref = db.ref(key);
     //console.log("getting key to", key)
       ref.once("value", function(snapshot) {
            console.log("the key is : ", key)
         console.log("the value is: ", snapshot.val());
         done(null, null);
      }, function (errorObject) {
   console.log("The read failed: " + errorObject.code);
   done(errorObject);
 });
    }
},


/**
 * Write nodes to the DB
 *
 * @param {string}   key   The key for the node
 * @param {object}   node  The full node with metadata
 * @param {function} done  Called when the node has been written
 *
 * @return {void}
 */

// need to basically code the gun functions myself

put: function(key, node, done) {
   /* if (this.initialized) {
```

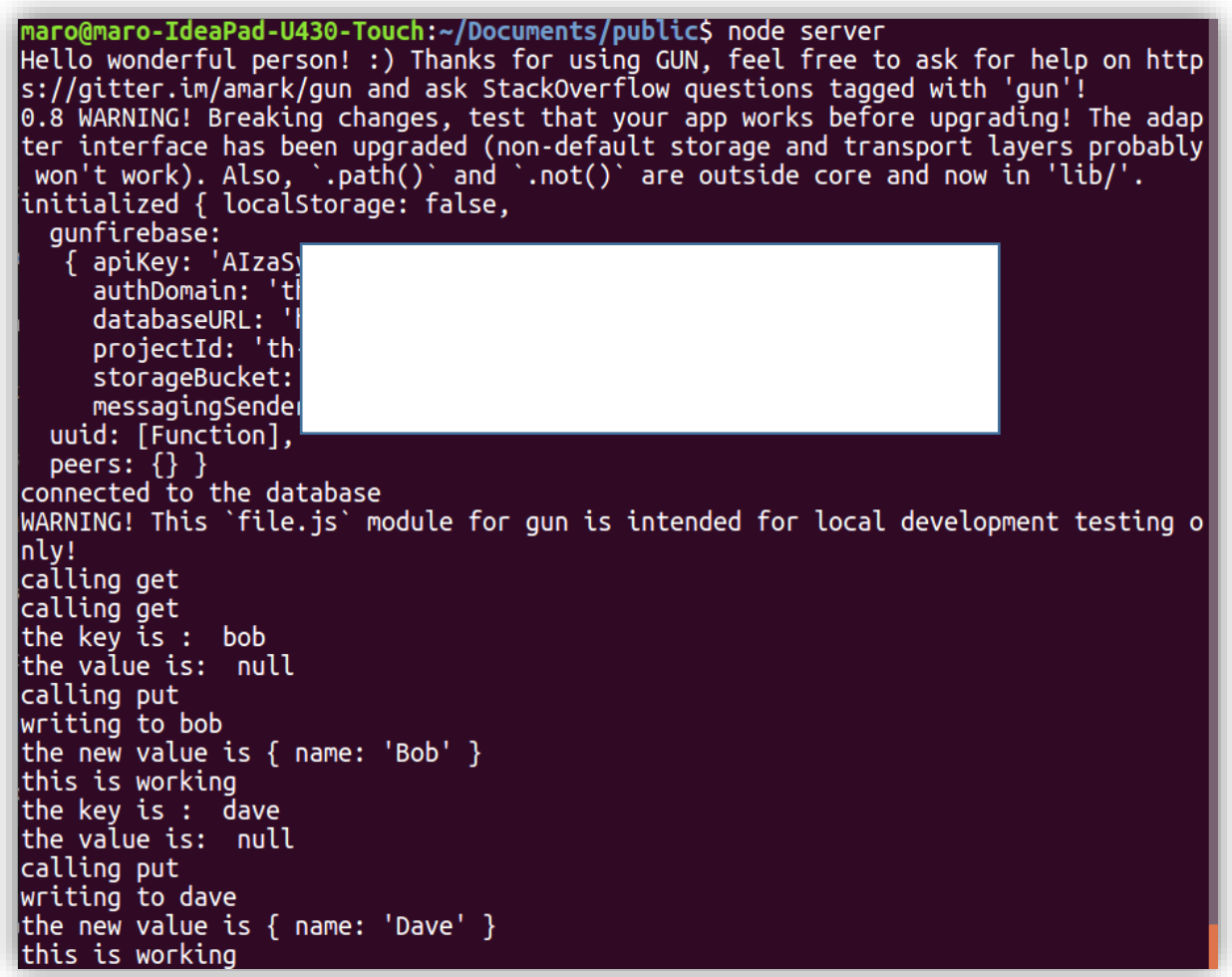
Screenshot of the gun-firebase adapter part 2

```
     *
     * @param {string}   key   The key for the node
     * @param {object}   node  The full node with metadata
     * @param {function} done  Called when the node has been written
     *
     * @return {void}
     */

     // need to basically code the gun functions myself

    put: function(key, node, done) {

       if (this.initialized) {
                console.log("calling put")

        delete node["_"]
        delete node[">"]
        delete node["#"]

            var db = firebase.database();
            var ref = db.ref();
            //var x = ref.child("th-yeat-project");

            var postsRef = ref.child(key);
            console.log("writing to", key)
            postsRef.set(
                 node
                , done())
                        console.log("the new value is", node)
                         console.log("this is working")

            }
   },

});
module.exports = GunFirebase
```

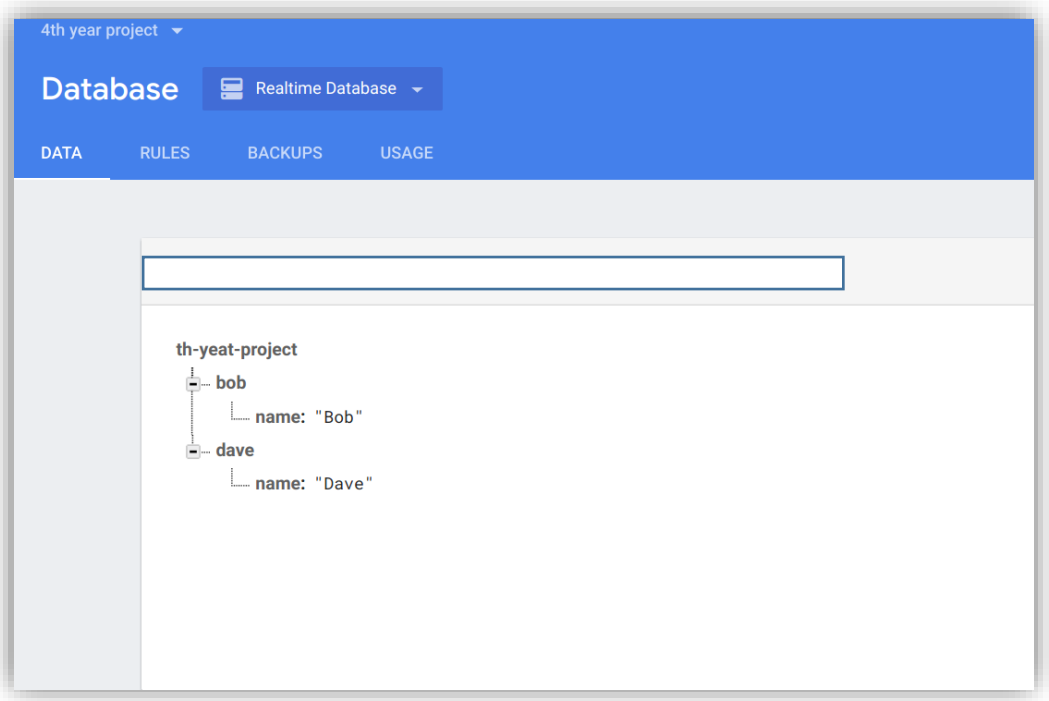
Screenshot of the gun-firebase adapter part 3

Appendix3 The gun-firebase adapter in action screenshots

Firebase initialising on server.js



The adapter working on the console



The data being entered the firebase console

Appendix4 Firebase real-time database limitations

Global

Operation	Limit	Description
Simultaneous connections	100,000	A simultaneous connection is equivalent to one mobile device, browser tab, or server app connected to the database. This isn't the same as the total number of users of your app, because your users don't all connect at once. For example, apps with 10 million monthly active users usually have fewer than 100,000 simultaneous connections. Your max simultaneous connections depends on your total user count and the average time users spend in your app. However, if you need to scale beyond this limit, try using multiple databases.
Simultaneous responses sent from a single database.	~100,000/second	Responses include simultaneous broadcast and read operations sent by the server from a single database at a given time. The limit refers to the data packets that represent each individual read or broadcast operation, including push notifications, sent from the database.
Number of Cloud Functions triggered by a single write	1000	While there isn't a limit to how many read or write operations you can trigger from a single function, a single database write operation can only trigger 1000 functions, per individual write operation. Cloud Functions can only be triggered by write operations, and each function can also trigger more write operations that trigger more functions (each with their own 1000-function limit).
Size of a single event triggered by a write	1 MB	The size of an event consists of the following values: 1. The existing data at the write location. 2. The update value, or the delta in data necessary to write the new data to the location. Write operations larger than 1MB succeed on the database, but they don't trigger a function invocation.
Data transfer to Cloud Functions	10MB/sec sustained	The rate of event data that can be forwarded to Cloud Functions.

Firebase Limitations listed on the Firebase documentation part 1

Data tree

Property	Limit	Description
Maximum depth of child nodes	32	Each path in your data tree must be less than 32 levels deep.
Length of a key	768 Bytes	Keys are UTF-8 encoded and can't contain new lines or any of the following characters: . $ # [ ] / or any ASCII control characters (0x00 - 0x1F and 0x7F)
Maximum size of a string	10 MB	Data is UTF-8 encoded.

Reads

Description	Limit	Notes
Size of a single response served by the database	256 MB	The size of data downloaded from the database at a single location should be less than 256 MB for each read operation. To perform a read operation at a larger location, try one of the following options: • Use a backup. • Paginate data with a query. • Use shallow queries.
Total nodes in a path with listeners or queries on it	75 million*	You can't listen to or query paths with more than 75 million nodes, cumulative. However, you can still listen to or query child nodes. Try drilling down deeper into the path or creating separate listeners or queries for more specific portions of the path. **You can't view paths with more than 30,000 total nodes from the data viewer in the Firebase console.**
Length of time a single query can run	15 minutes*	A single query can run for up to 15 minutes before failing. **A single query performed in the Firebase console can only run for up to 5 seconds before failing.**

Firebase Limitations listed on the Firebase documentation part 2

Writes

Description	Limit	Notes
Size of a single write request to the database	256 MB from the REST API; 16 MB from the SDKs.	The total data in each write operation should be less than 256 MB. Multi-path updates are subject to the same size limitation.
Bytes written	64 MB/minute	The total bytes written through simultaneous write operations on the database at any given time.

Firebase Limitations listed on the Firebase documentation part 3

Appendix 5 Results discussion (Circular reference)

```
const bob = gun.get('bob').put({ name: 'Bob' })
const dave = gun.get('dave').put({ name: 'Dave' })

// Write circular reference.
bob.get('friend').put(dave)
dave.get('friend').put(bob)

// Print the data!
bob.get('friend').val(friend => {
    console.log(friend.name) // Dave
});

// Now with a circular reference
bob.get('friend').get('friend').val(friend => {
    console.log(friend.name) // Bob
});
```

Screenshot of the circular reference code

```
calling get
calling get
calling get
calling get
calling get
calling get
calling get
calling get
Dave
Bob
the key is :  bob
the value is:  { name: 'Bob' }
the key is :  bob
the value is:  { name: 'Bob' }
calling put
writing to bob
the key is :  bob
the value is:  { name: 'Bob' }
calling put
writing to bob
the key is :  bob
the value is:  { name: 'Bob' }
calling put
writing to bob
the new value is { name: 'Bob' }
this is working
the key is :  dave
the value is:  { name: 'Dave' }
the key is :  dave
the value is:  { name: 'Dave' }
calling put
writing to dave
the key is :  dave
the value is:  { name: 'Dave' }
calling put
writing to dave
the key is :  dave
the value is:  { name: 'Dave' }
calling put
writing to dave
the new value is { name: 'Dave' }
this is working
FIREBASE WARNING: Exception was thrown by user callback. Error: Reference.set failed: First argument  contains an invalid key (#) in property 'bob.bob'.  Keys must b
e non-empty strings and can't contain ".", "#", "$", "/", "[", or "]"
```

Screenshot of the error in the console