Advanced database programming Continual Assessment (2020)

Implementing CouchDB on a Web based application

James Mahoney

I.T Tralee, Tralee, Co. Kerry, Eire.

james.mahoney@students.ittralee.ie

**Abstract**

The purpose of this paper is to develop a web-based application that uses Apache CouchDB as a database and to explore CouchDB itself and the various JavaScript drivers for CouchDB. This paper will outline the steps to take to develop such an application and will have examples of code and screenshots of code in use. The CRUD functionalities of CouchDB and the Map-Reduce function and views will be explained with screenshots and examples of code in action. The Code Written during this paper is available at <https://www.github.com/mahoney0101/buglog> .

*Keywords:* couchdb; pouchdb; nano; cradle; web application;

1. Introduction

The aim of this paper is to investigate CouchDB as a database technology and to investigate the different JavaScript drivers for the software. Once a comprehensive comparison has been completed develop a web application with a chosen CouchDB driver. The web application will use ReactJS as a front-end technology and CouchDB as the accompanying database technology using an appropriate JavaScript driver for the CouchDB framework.

1. Research Method

Research for this paper was performed by viewing documentation for CouchDB and documentation for different java script CouchDB drivers and bringing the information together to get the different aspects of this technology along with becoming familiar with the Technology itself from a hands on perspective.

1. CouchDB.

CouchDB is a JSON document oriented No SQL database implemented in the programming language Erlang that is open source. It was created by Damien Katz in April of 2005 after two years of self-funded work. He later open sourced the software and released it to the public under the General Public License. CouchDB uses JavaScript as its main language to query using Map-reduce functions. CouchDB stores its data in collections of non-relational documents. Each of these documents are independent and there is no schema required.

CouchDB can be defined as (Juravich, 2012)

* A document database server, accessible via a RESTful JSON API.
* Ad-hoc and schema free with a flat desirable space
* Distributed featuring robust and incremental replication with bi-directional conflict detection and management.
* Query-able and index-able, feature a table-oriented reporting engine that uses JavaScript as a query language.

* 1. Features of CouchDB

The motivation behind CouchDB’s development can be defined with one word: relax. CouchDB comes with a suite of features designed to lower the effort of running resilient distributed system. Here are some key features of CouchDB and how it is different from other No SQL databases. (apache-couchdb, 2020)

* + 1. Replication

A defining feature of CouchDB is its ability of bi-directional replication. This type of replication allows a developer to synchronise data between mobile devices or laptops. This feature means that the data is unlikely to be lost and it frees up computing power as if an application is offline the user of the application can still use the app as if it were online. The application will use its local database and when the option to sync becomes available the device can sync across servers. During the sync, the software can manage potential data clashes. The software will determine the most recent data so users will always have access to the most up to date data. CouchDB does not differentiate between data that is housed on either one or many servers. CouchDB can determine valid data that is closer to the user via geo-location. The consequence of this is that the user quickly has accurate recent data, and it is easy for CouchDB to manage databases.

* + 1. Views

Views are the main way that CouchDB runs queries. Views are built using the JavaScript programming language and they allow the user to find information. The information can be mapped and extracted in specific order the same as an SQL query would. Views are created and stored separately to the data inside the database as its own view. This makes querying a large database much quicker if the user is using a predefined view. Views can be replicated across databases just like the data and do not affect the database. Along with the view feature there is the Map-reduce functionality which allows the user to run the JavaScript map function on the gathered data and to manipulate each document.

* + 1. REST API

CouchDB implements a RESTful API. Which makes CouchDB “flexible, fast and powerful” (apache-couchdb, 2020)

* + 1. Built for offline use

CouchDB works well offline. CouchDB allows developers to develop applications to store data in a local database on the machine until the application connects to the internet allowing the local database to sync with the remote database.

* + 1. Efficient

CouchDB does not limit the amount of data in each element or the number of elements you can store in each document. Whereas in other relational databases there would be either an allocated size for a column or max size.

* + 1. Compatibility

CouchDB is compatible with many operating systems and has various drivers (PouchDB which was designed to run on mobile devices and be the middle-ware between CouchDB and a JavaScript application) which bridge the gap between applications of various programming languages, making it very easy to work with for developers.

1. CouchDB JavaScript drivers

There are many drivers for CouchDB to use with various languages. For the purpose of this project the light will be shown on JavaScript drivers. A JavaScript CouchDB driver is a JavaScript library written to communicate with the Database from the client. This can usually be installed via Node package manager and used within the application. These drivers make the whole process easier. It is essential that when a developer chooses a driver to use that they choose a driver that has a large user base, adequate documentation and recent updates. As there are drivers out there that exist but have no current support. If a driver library has no support, they can be victim to security threats or deprecated code. For the purposes of this project this paper will look at three CouchDB JavaScript drivers; Nano, Cradle and PouchDB.

* 1. Cradle

The last release of this driver is five years old (cradle, 2020) suggesting that there is no longer support for this driver. This would not be a good library to use in a project as there would be deprecated code and issues in the codebase.

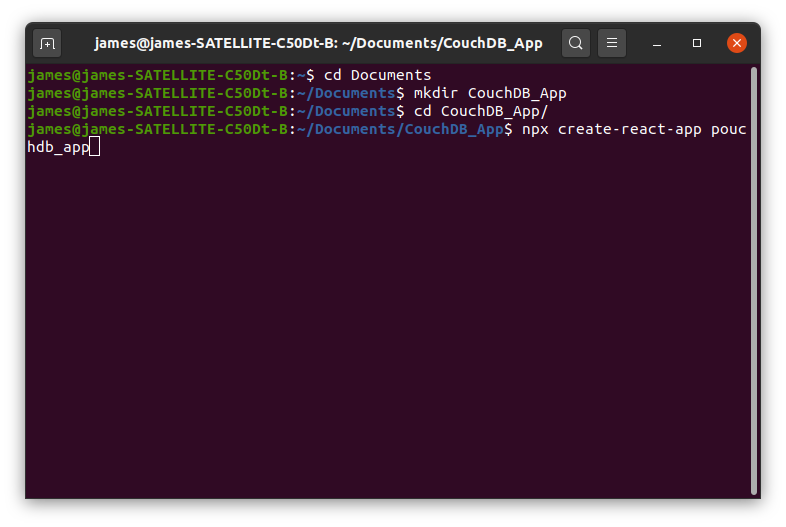
* 1. Nano

Five years ago, this would have been a great choice for a developer however the GitHub page for Nano shows multiple open issues that have not been closed. These issues are spanning back as far as five years and multiple ignored pull requests. This can be a bad sign where support is concerned. If a developer were to use this driver and there was a blocking issue there would be no support for the software and would be extra work for a developer to fix issue (apache/nano, 2020).

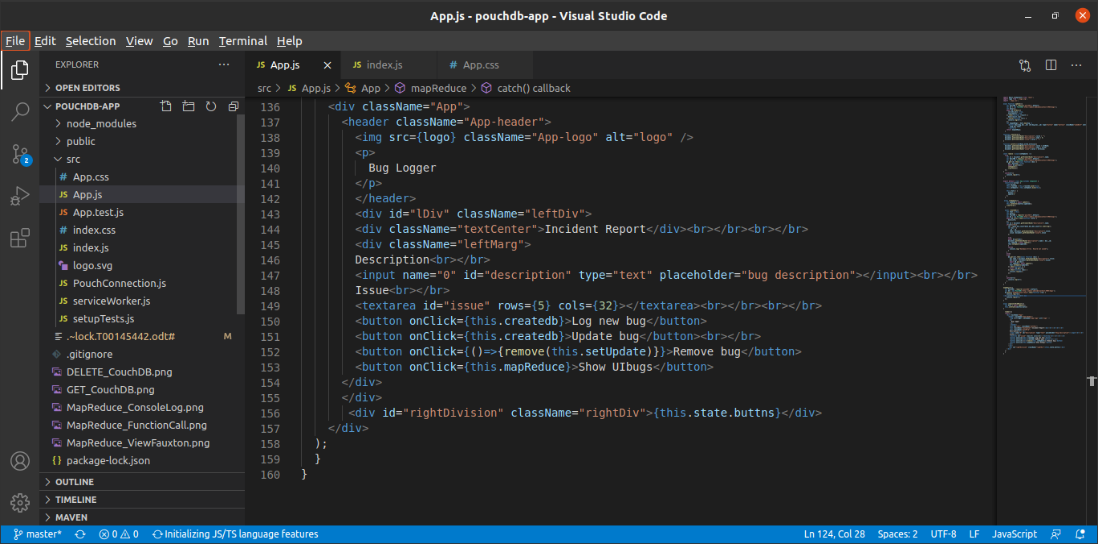
* 1. PouchDB

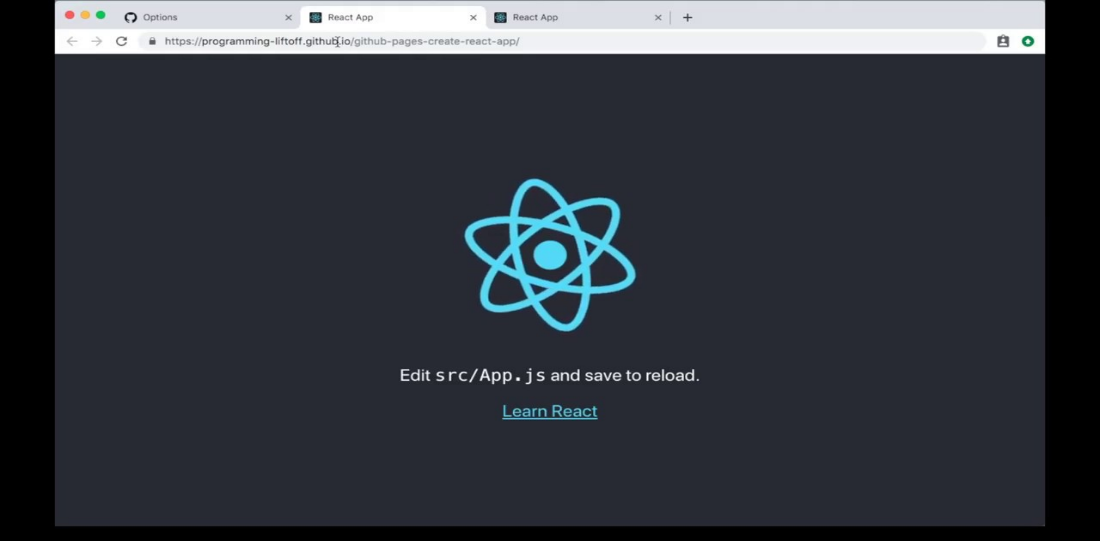
“PouchDB is an open-source JavaScript database inspired by [Apache CouchDB](http://couchdb.apache.org/) that is designed to run well within the browser. PouchDB was created to help web developers build applications that work as well offline as they do online. It enables applications to store data locally while offline, then synchronize it with CouchDB and compatible servers when the application is back online, keeping the user's data in sync no matter where they next login.”(About PouchDB, 2020)

1. Developing the application
   1. Creating the ReactJS UI

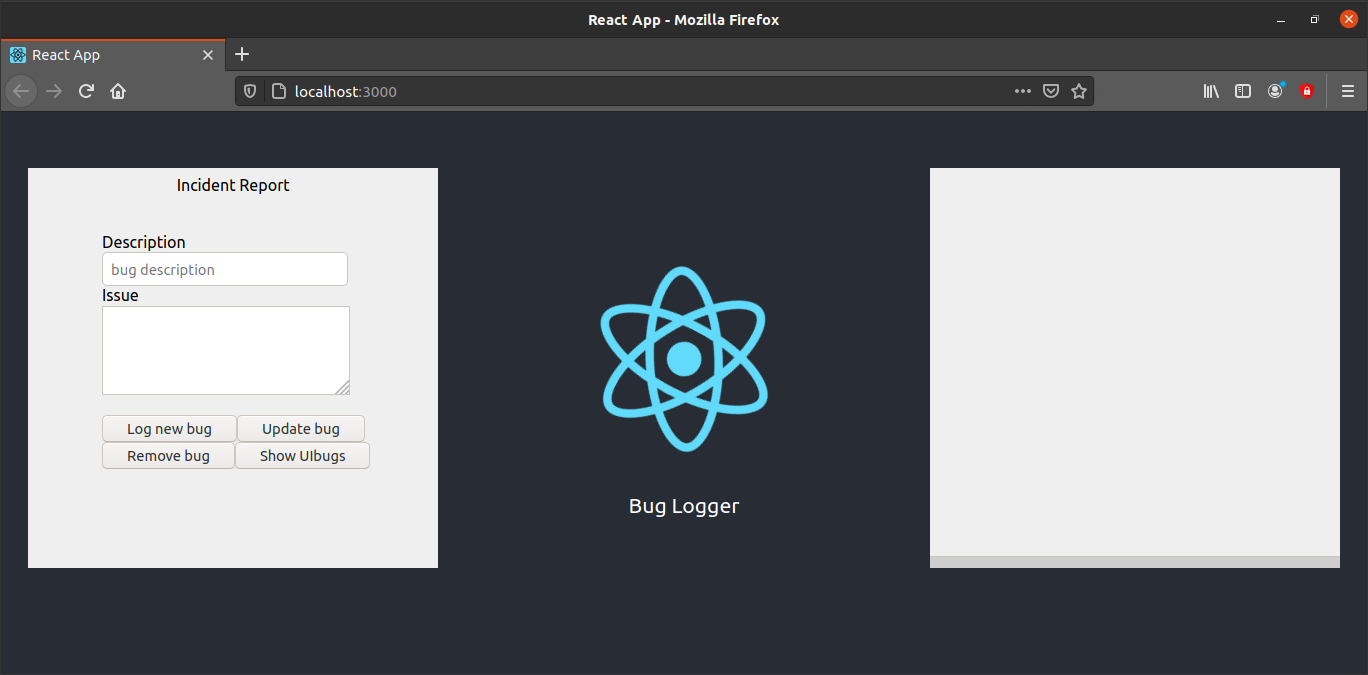
This application will be a web application and the core of the application will be created using ReactJS, PouchDB and CouchDB. First off to create the React front end of the application will be simple as we are not demonstrating user interface ability this will be a simple create-react-app command.

Once the application has been created the file structure will be as shown below. For the purposes of this demonstration the changes in code will all be placed into the App.js file. This is the main component of the application. The UI that the create-react-app creates as below.





The next step for the UI is to add two division(<div>) tags and create a division either side of the page. The left will hold the form for the create and update methods, along with buttons to call the CRUD function that will be need to create read update and delete records in the CouchDB database that will be connected to the application and a button to call the Map-Reduce function as seen below.



* 1. Creating the CouchDB database

To install CouchDB on a Debian machine, first install the repository key in terminal via:

$ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys \

8756C4F765C9AC3CB6B85D62379CE192D401AB61

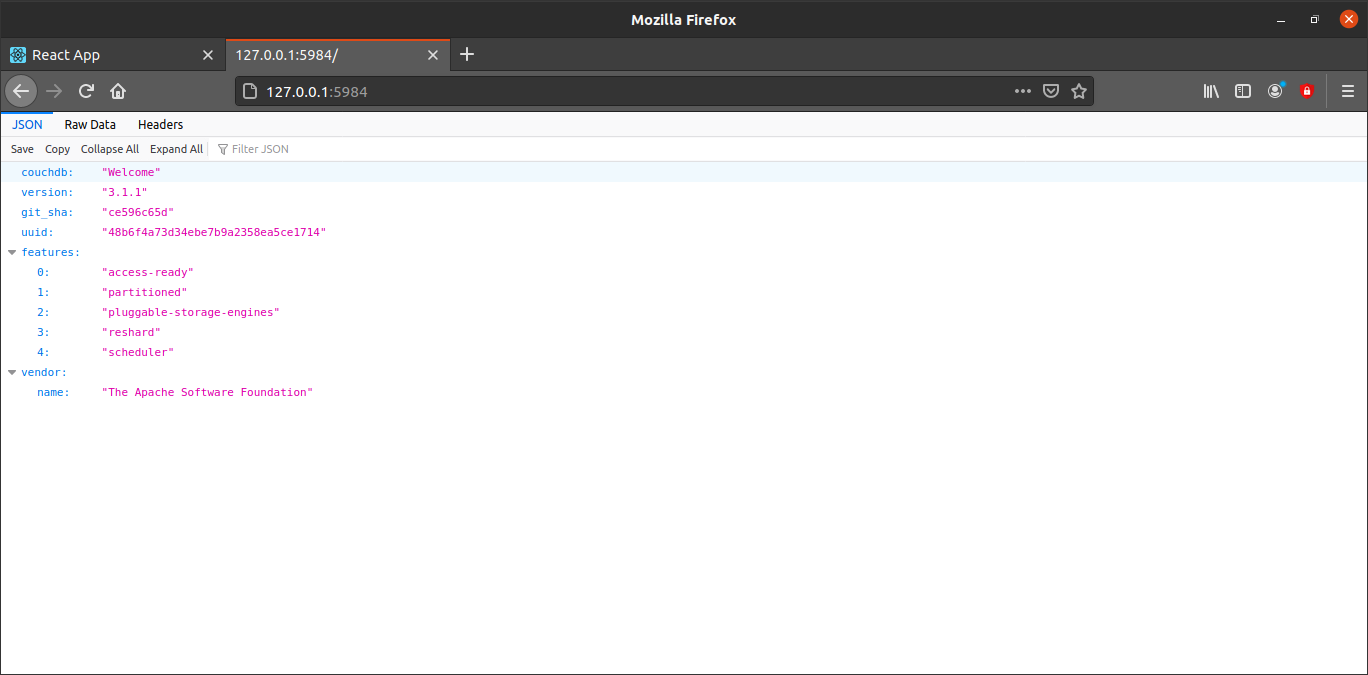
Next update the repository cache and install the package:

$ sudo apt update

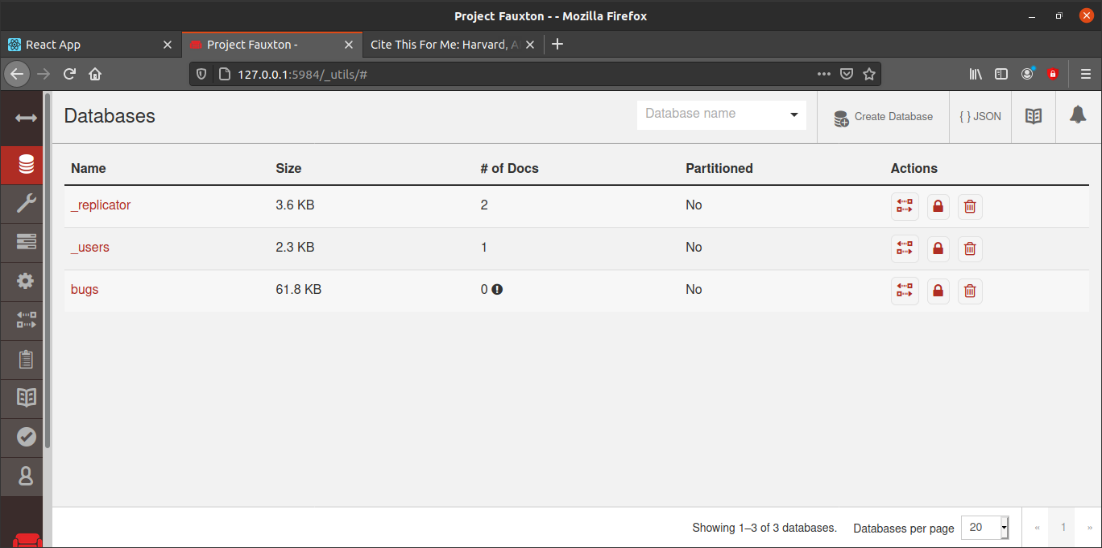
$ sudo apt install -y couchdb

(1.1. Installation on Unix-like systems — Apache CouchDB® 3.1 Documentation, 2020)

Once the package has been installed to make sure the package has been installed and is functional enter http://127.0.0.1:5984 into the address bar and the response should be as shown here.

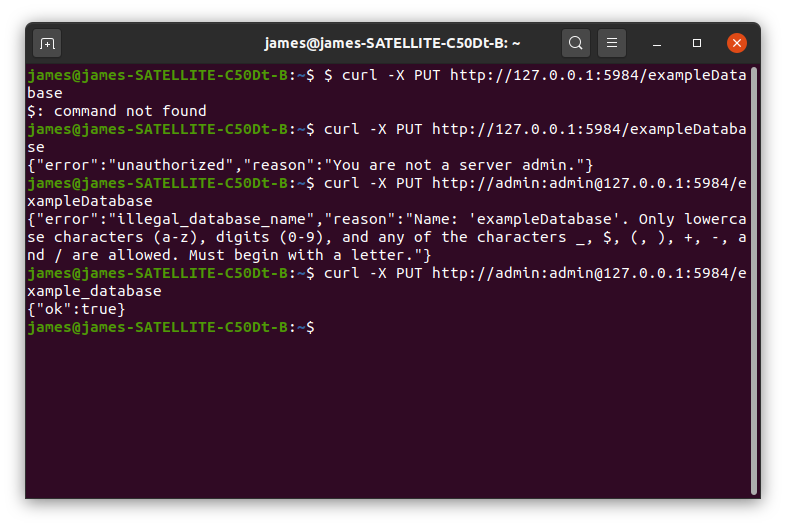


Once the installation is confirmed the in built graphical user interface Project Fauxton is accessible via the address bar in the browser at the address <http://127.0.0.1:5984/_utils/># .



Now that there is access to the GUI and the software is installed a database can be create by using the curl command $ curl -X PUT http://127.0.0.1:5984/databaseName

If there is a username password needed, enter if before the IP address as http://admin:admin@127.0.0.1/databaseName**.** CouchDB should send confirmation in the form of {"ok":true}.

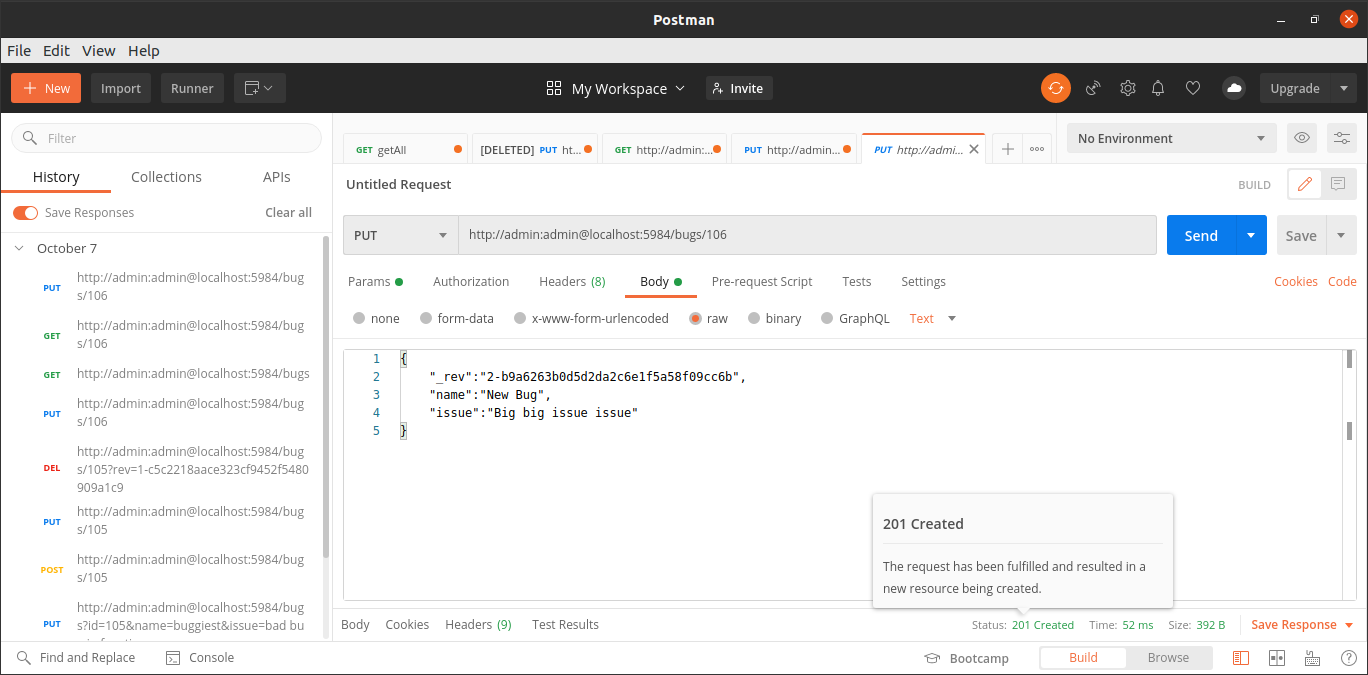


* 1. Postman requests

For further confirmation that the database is in working order Postman can be used to show examples of CRUD on the database.

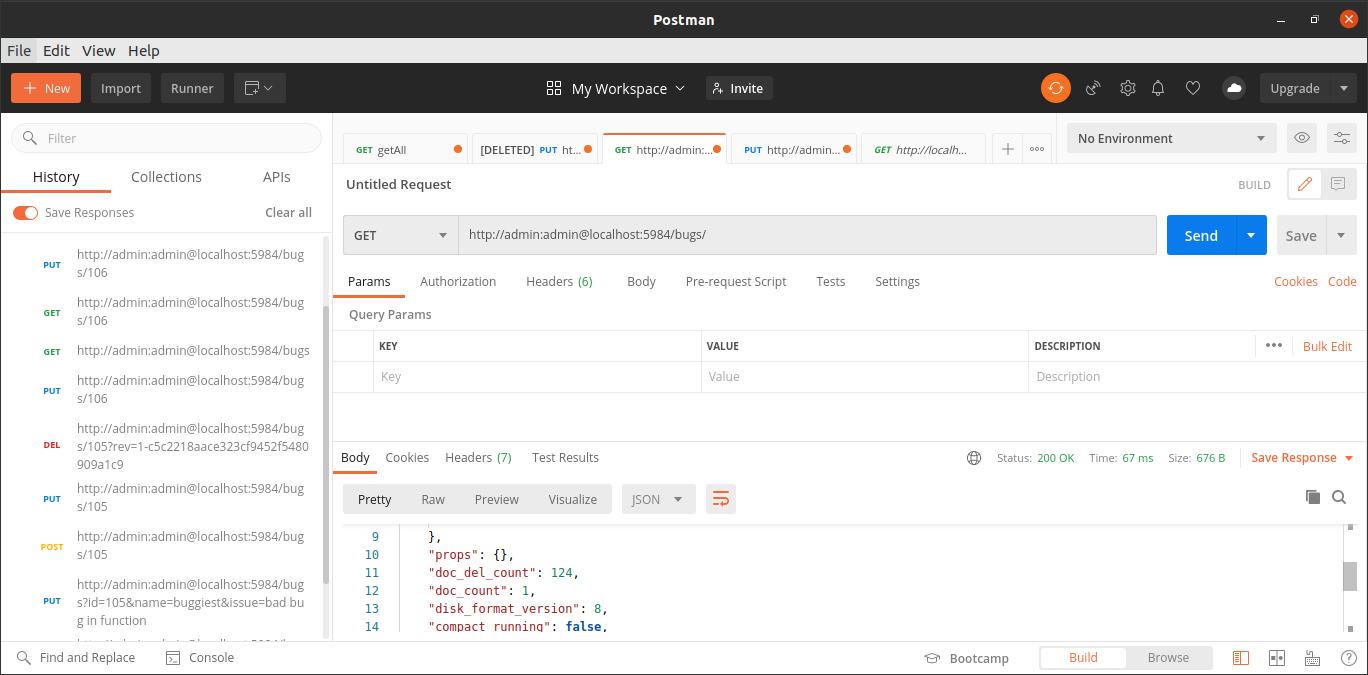
* + 1. Create

The PUT request to the “bugs” database returned a 201 created response signaling that the resource has been created in the database.



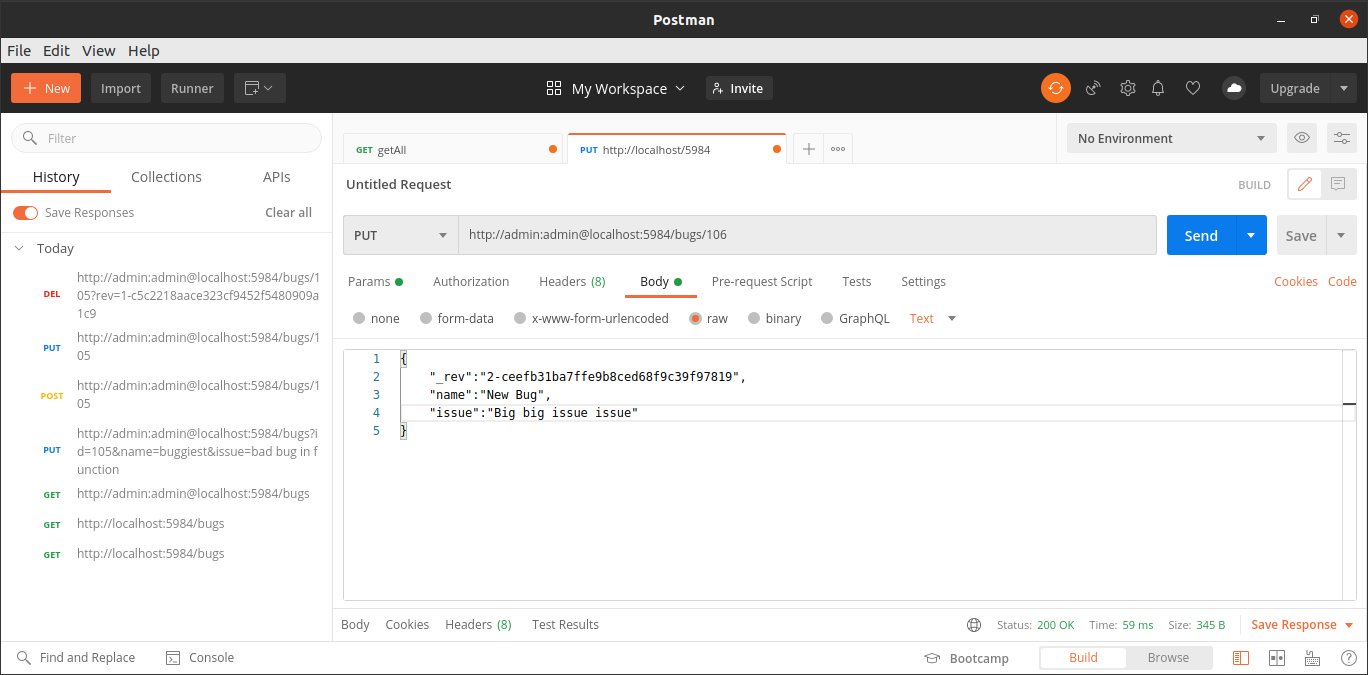
* + 1. Read

The GET request returned a 200 OK status signaling that the request was performed with no error. The request returned the contents on the database. At the time of the request there was one document in the database.

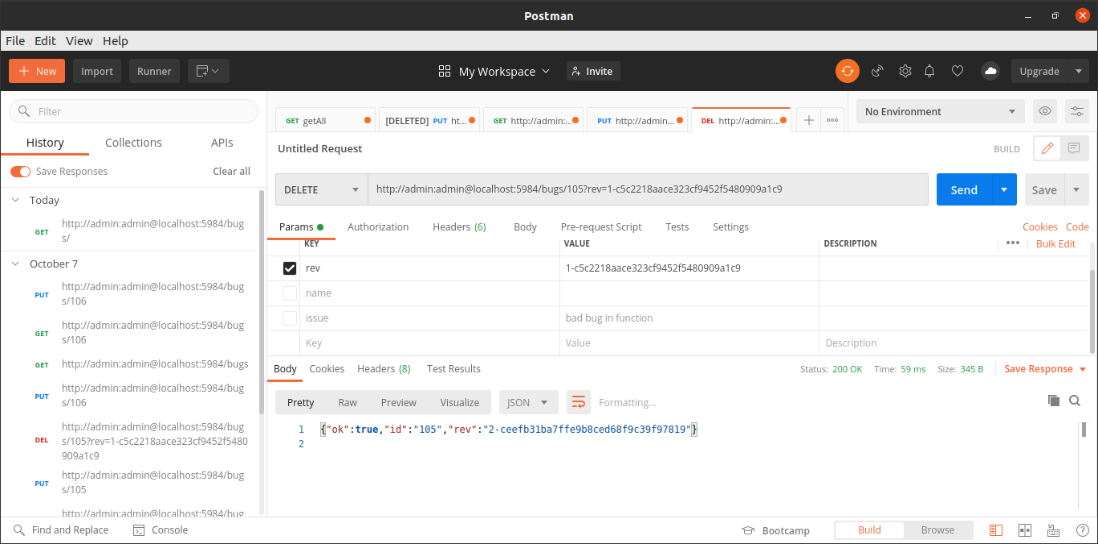


* + 1. Update

The PUT request returned a 200 OK status meaning that the document was updated. The \_rev must be in the body of this request. To get the current revision of the document first there must be a get performed on the document then return the PUT request with the current revision and the updated document. CouchDB will return the 200 OK status along with the new \_rev

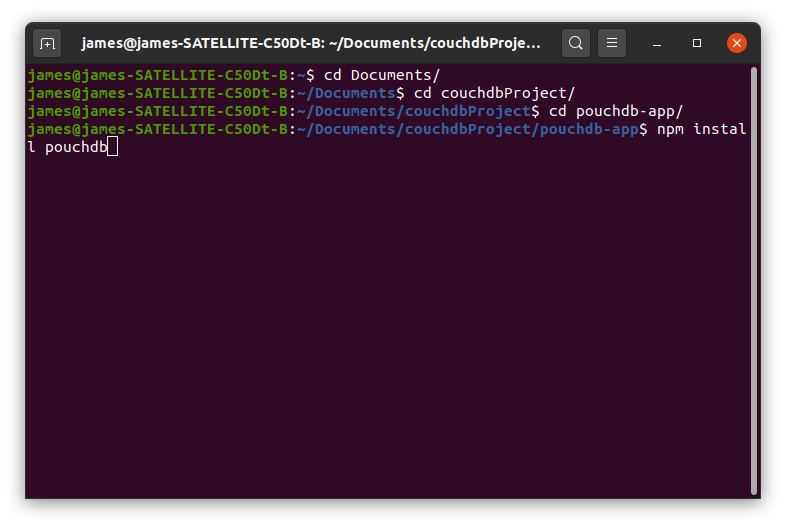


* + 1. Delete

The DELETE request returned a 200 OK status. The \_rev must be included in this request also and the database returns the new revision of the document

* 1. PouchDB

To install PouchDB in the application through Node package manager run the change directories into the directory where the application is and run the command “npm install pouchdb”.



Once the package is installed PouchDB can be used in the application by using the code:

var PouchDB = require('pouchdb').default;

var db = new PouchDB('http://admin:admin@localhost:5984/bugs');

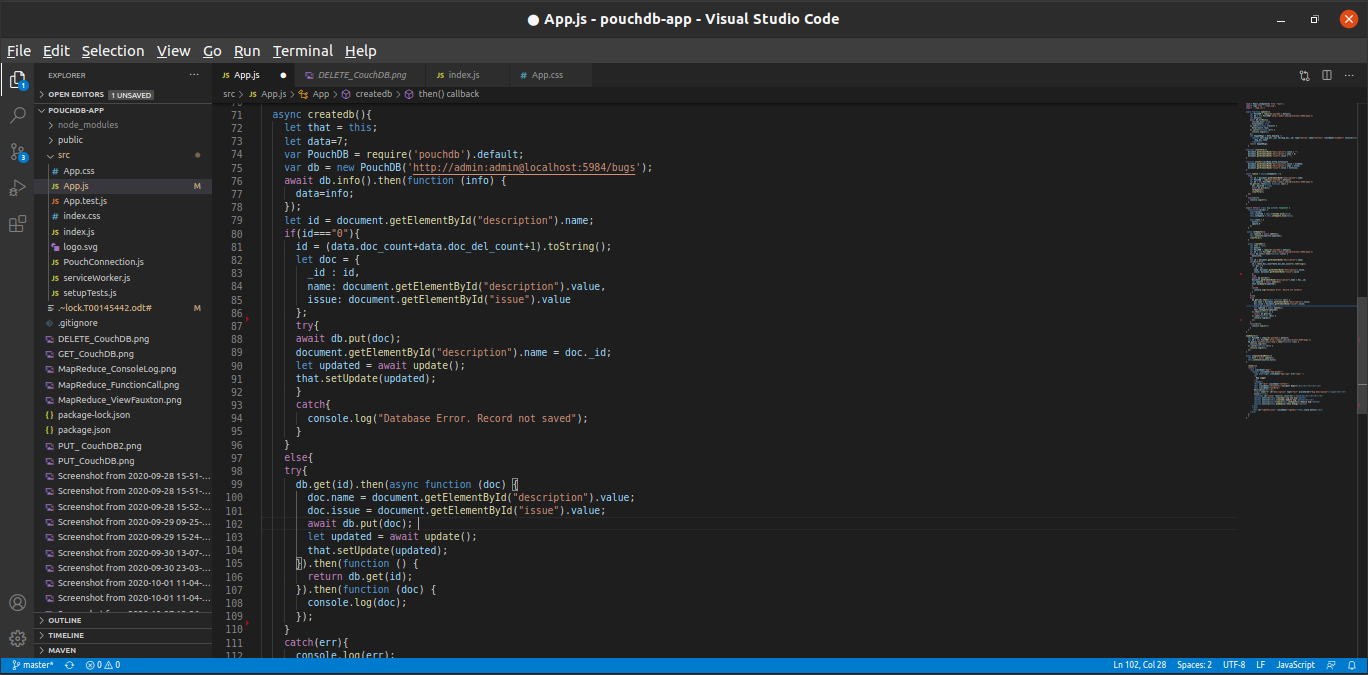
* + 1. Create/Update

The first line of code constructs the PouchDB object by requiring the pouchdb package and the next line creates an instance of a database names “bugs”. If the database already exists, then the object points to the existing database.

Now that the connection is made CRUD calls can be made to and from the database. An example of the createdb. The code will decide whether to create or update whether there is an item selected in UI.

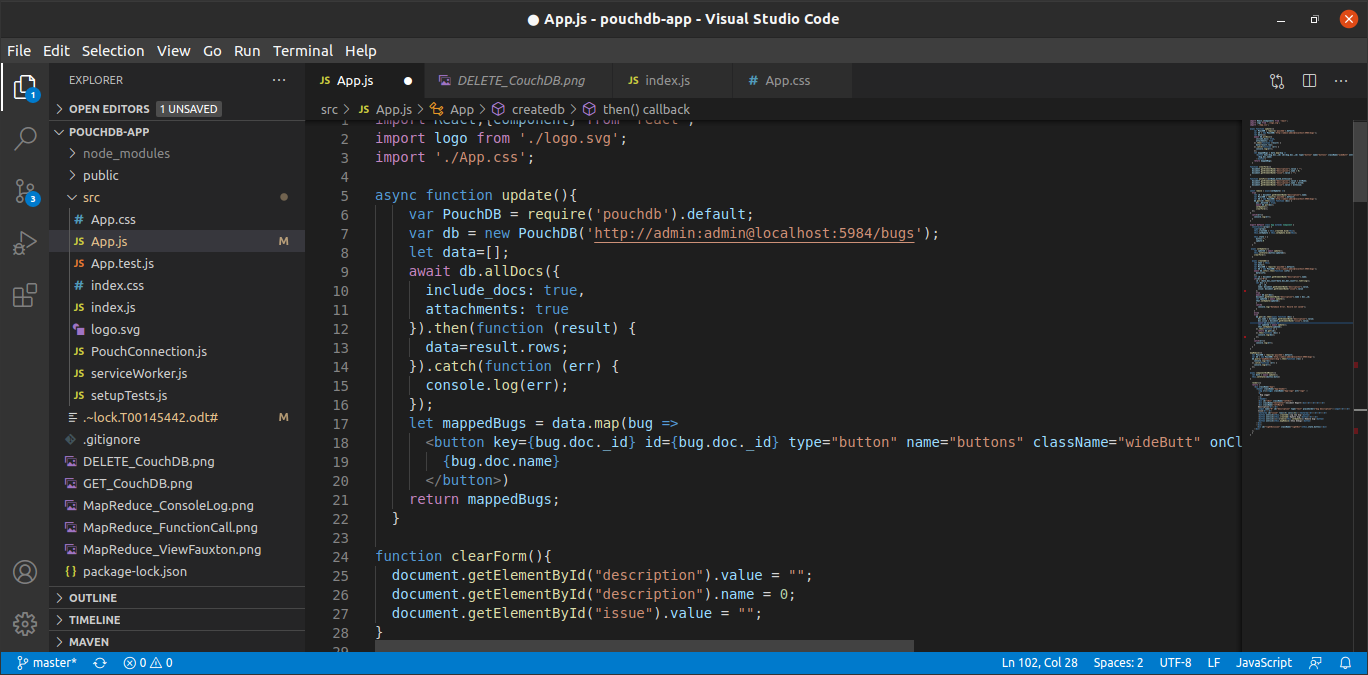
This code will:

* Connect to the required databases.
* Check if the id in the UI is 0 or not 0 (if its zero it is a blank form, not zero then an item has been selected).
* With the result of the previous check the code will either:
* Create the document and create it in the database.
* Get the previous version of the selected document and update it using the new information and the previous revision number.
* If there are any error, it will log those errors to the console.



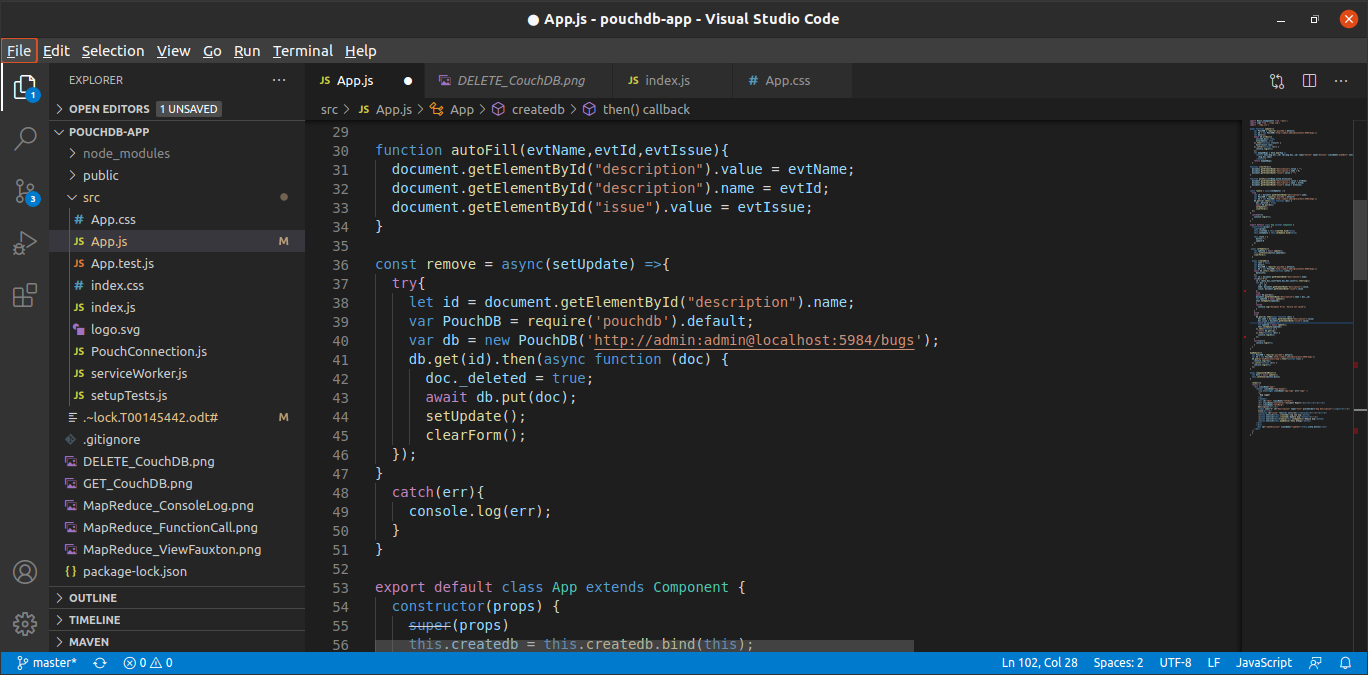
* + 1. Read

An example of a read function can be seen below. This is an asynchronous function that makes a call to the database and returns all the documents. Once the documents are present there is manipulation in the UI.



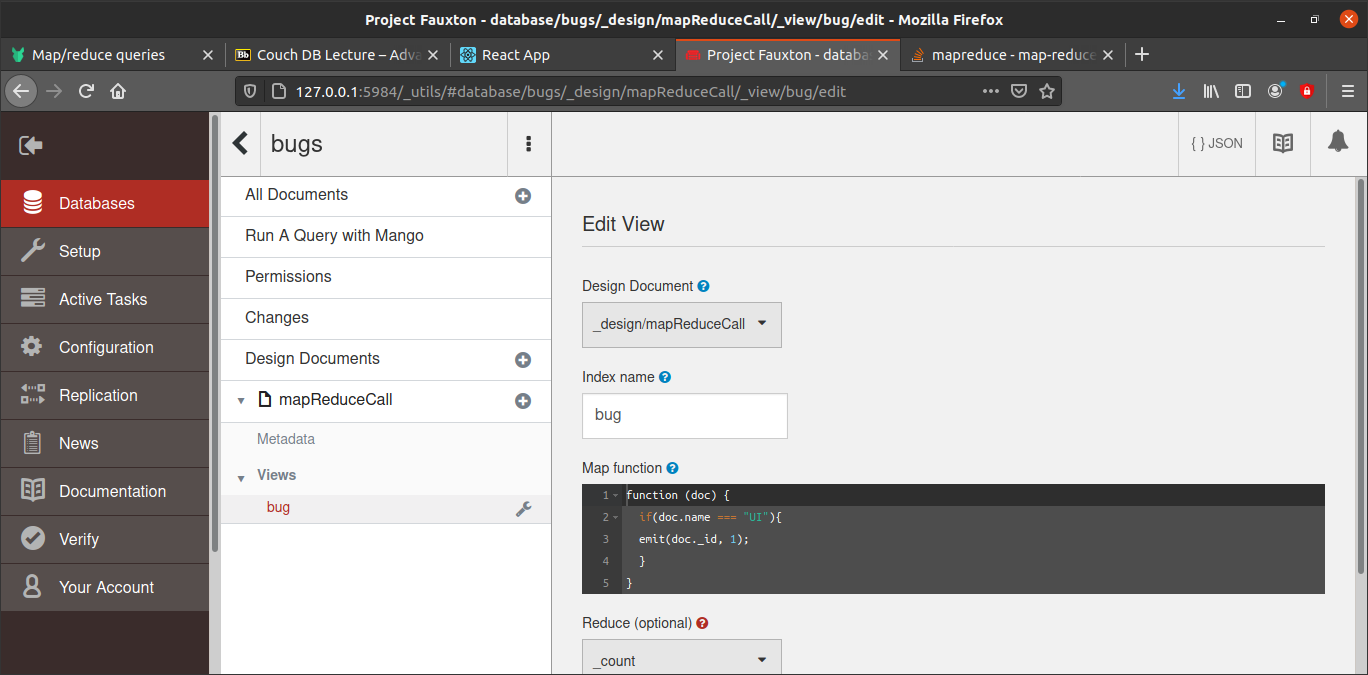
* + 1. Delete

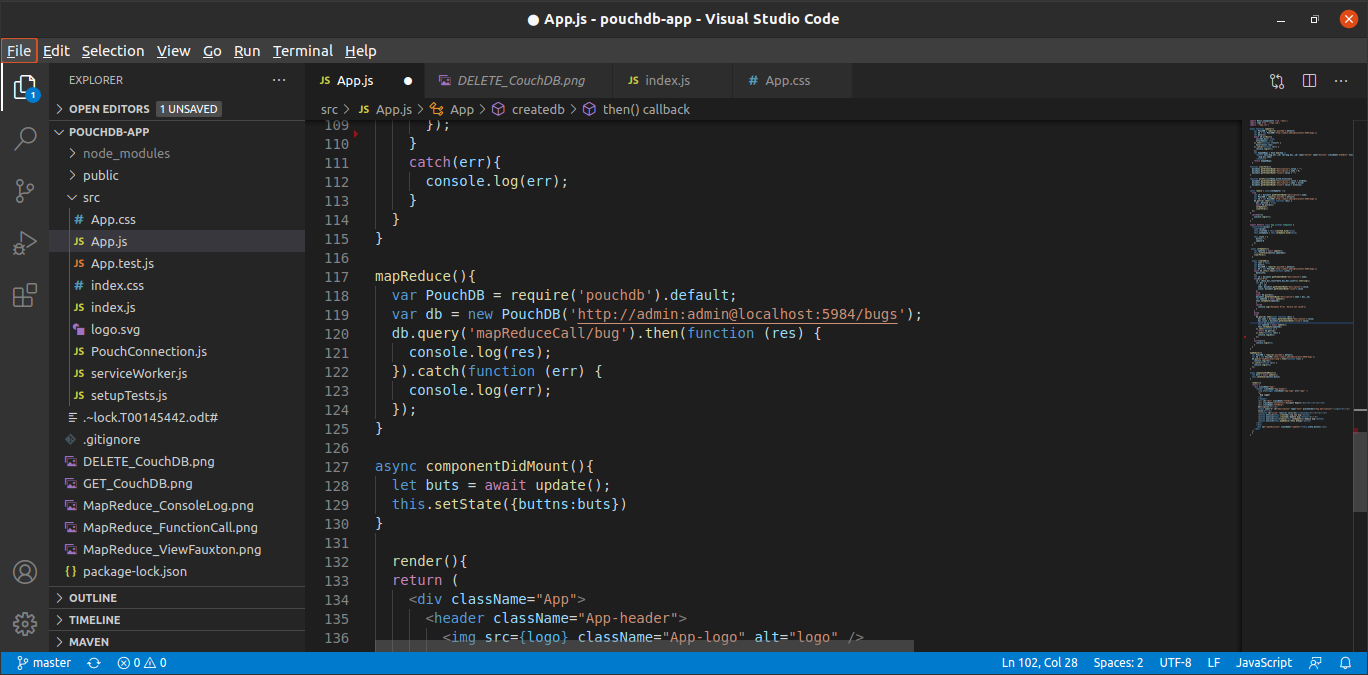
The code below (Figure 14) is an example of a delete function using PouchDB. This code retrieves the current version of the selected document and sets deleted to true. This will delete the document from the database. Once deleted is set to true the code sends the now “deleted” document back to CouchDB



* + 1. Map-Reduce

Below is an example of how to create a view in Project Fauxton interface. Once the View has been created PouchDB can call this from the application.

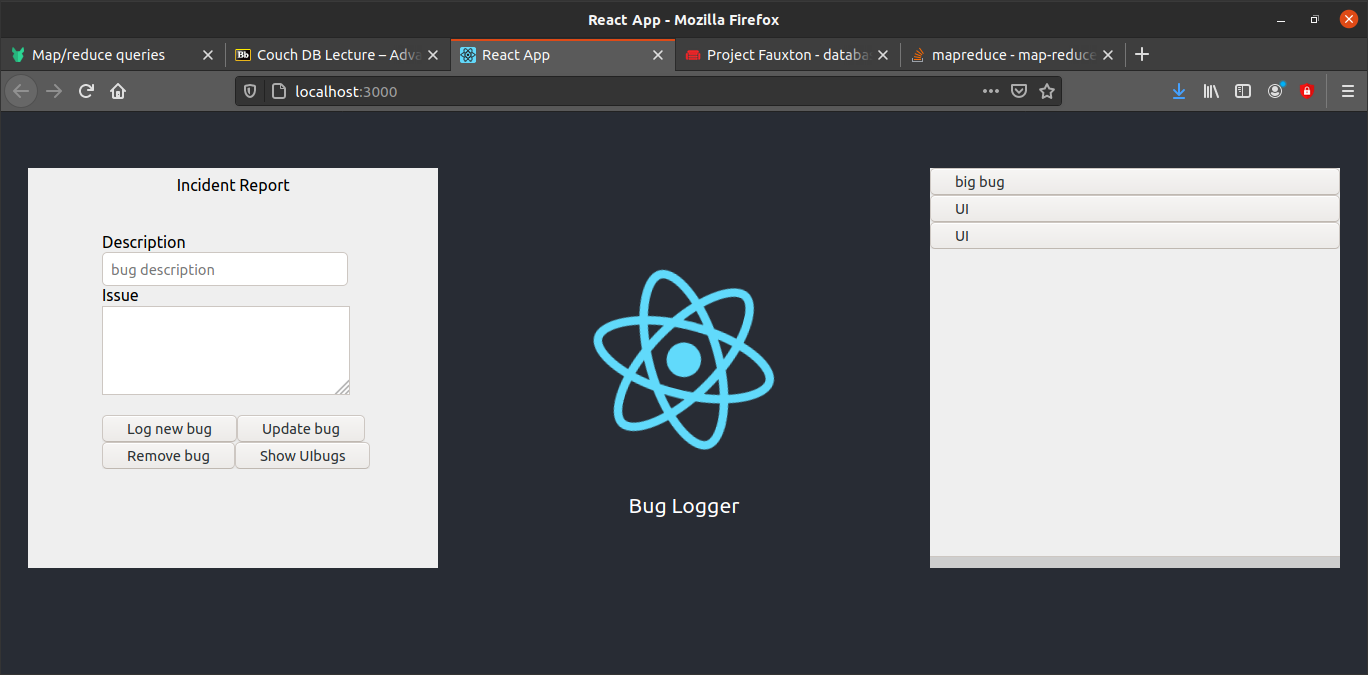




This Map-Reduce function will return the number of documents that have the value of the “name” key set to “UI”. The application user will be able to see how many bugs have been logged for the UI.

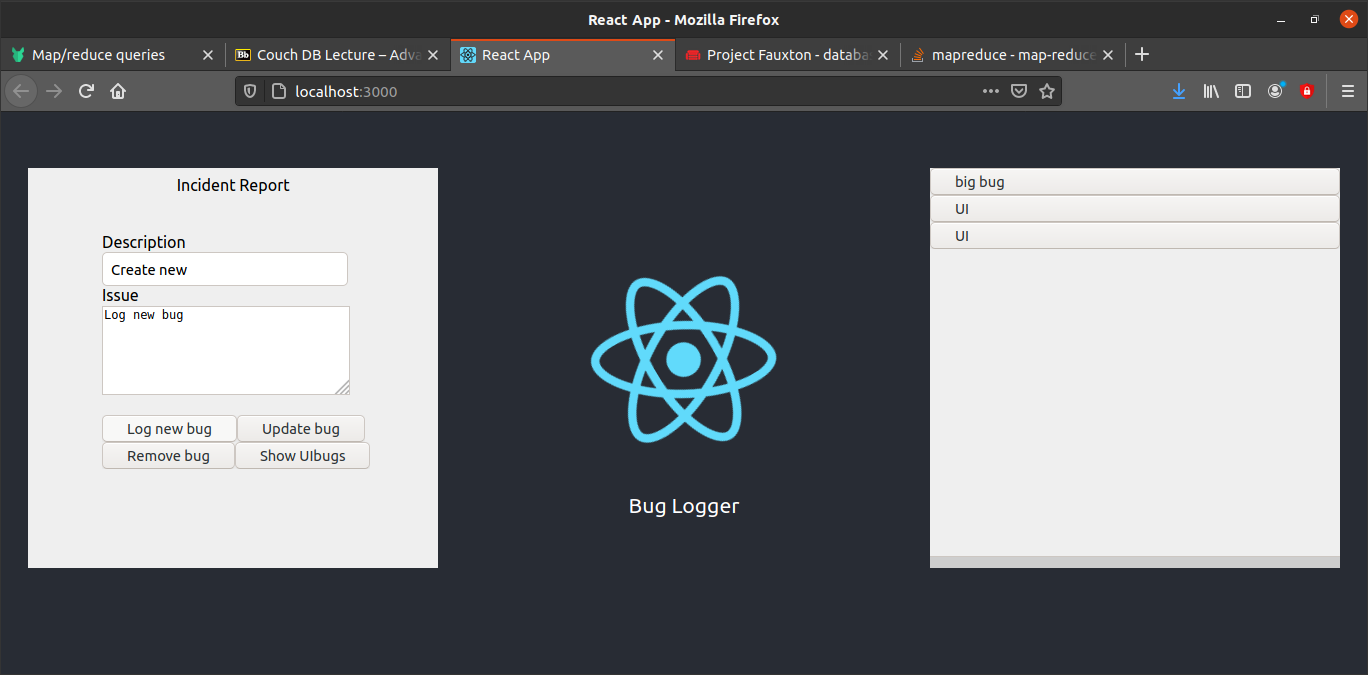
1. Finished application

The application is now functional and ready to be deployed. It has CRUD functionality and communicates with a CouchDB database via PouchDB.



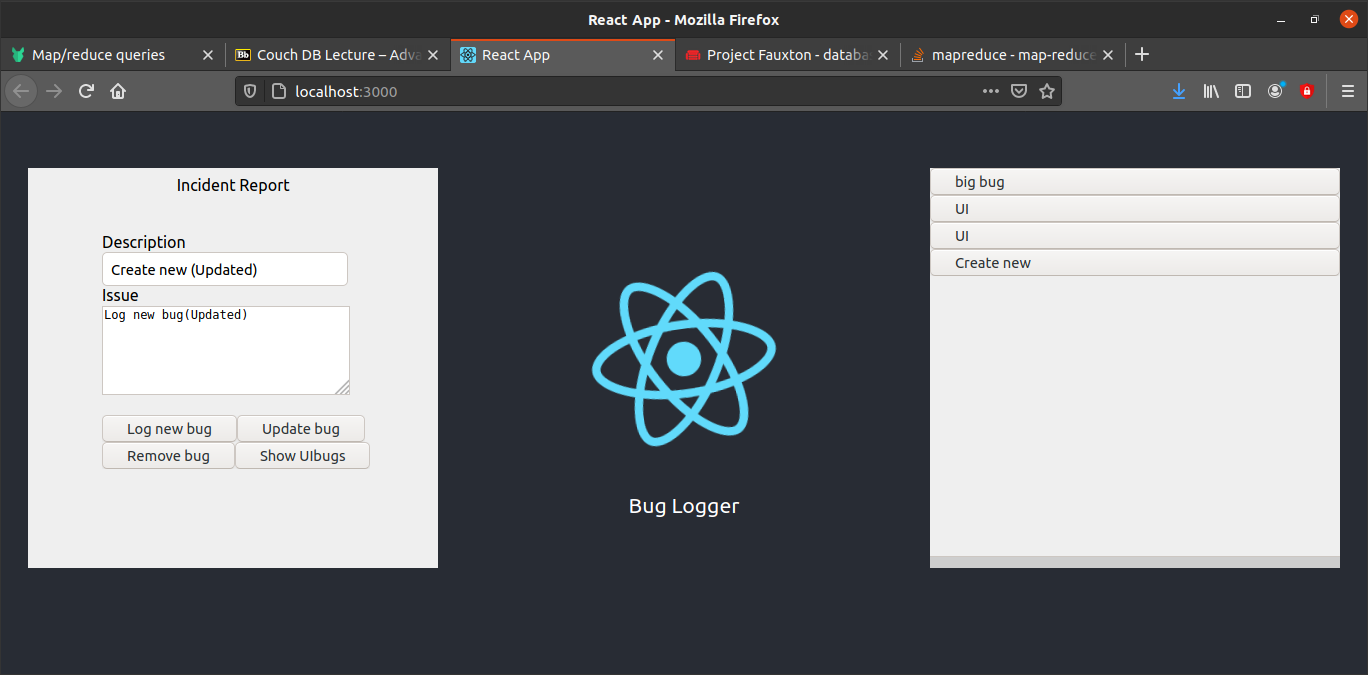
* 1. Create

To log a new bug just enter the information in the form and select the “Log new bug” button.



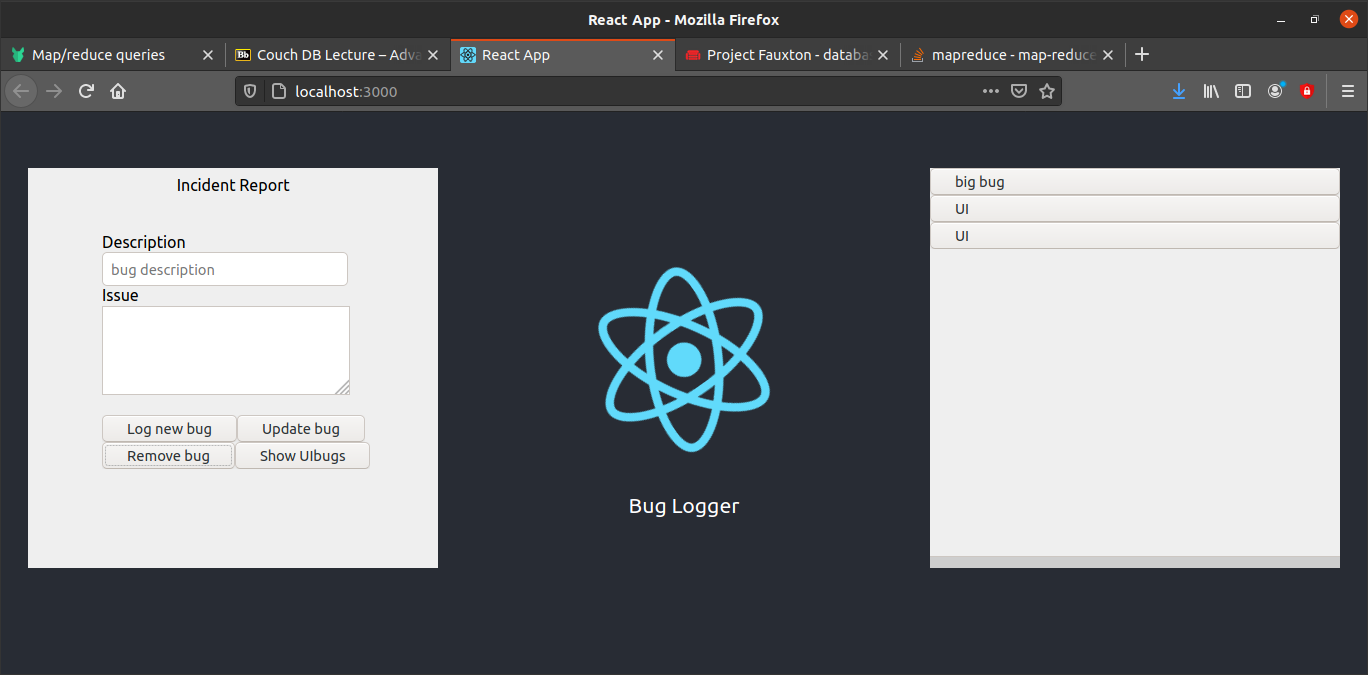
* 1. Update

To update the existing bug the bug from the list on the right-hand side and edit it in the form. Once edited select the “Update bug” button.



* 1. Delete

To delete a document, select the bug from the right division and once its selected click the “Delete” button.



* 1. Read

The read function is called in the ComponentDidMount() function when the app component loads. This is how the list of bugs is generated.

1. Conclusion

During the development of this application it became very clear that although there may be many JavaScript drivers out there for CouchDB the reliability of these sources is questionable. It seems that during the time when CouchDB was freshly developed there were many developers creating driver libraries. Almost all these libraries have been abandoned. PouchDB is by far the most up to date and supported JavaScript driver library. Therefore, it is certainly the correct choice for any developer who wishes to develop any type of a JavaScript based application and use CouchDB as a database.

It is obvious why CouchDB’s redundancy is popular with so many larger companies such as Verizon, Ancestry.com and others. Since CouchDB saves each revision of the database it is easy to keep track of the history in the data, however if storage space is a problem then CouchDB may not be the right choice for a project.

References

Jain, R., Singh, S.K. and Mishra, B., 2019. A Brief Study on Build Failures in Continuous Integration: Causation and Effect. In Progress in Advanced Computing and Intelligent Engineering (pp. 17-27). Springer, Singapore.

Volker Gruhn Christoph Hannebauer Christian John(2013). Security of Public Continuous Integration Services

Vasilescu, B., Yu, Y., Wang, H., Devanbu, P. and Filkov, V. (2019). Quality and Productivity Outcomes Relating to Continuous Integration in GitHub.

Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. 2016. Usage, Costs, and Benefits of Continuous Integration in Open-Source Projects. In ASE.

Noureddine Kerzazi and Bram Adams. 2016. Botched Releases: Do We Need to Roll Back? Empirical Study on a Commercial Web App. In SANER

Nelson, N., Hilson, M., Tunnel, T., Marinov, D. and Dig, D. (2017). Trade-offs in Continuous Integration: Assurance, Security, and Flexibility.

Ieeexplore.ieee.org. (2019). Continuous Integration and Its Tools - IEEE Journals & Magazine. [online] Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6802994 [Accessed 2 Dec. 2019].