

CouchDB

Review It

Mark | Advanced Database Programming| 31/10/19

Table of Contents

[Title 4](#_Toc24390891)

[Summary 4](#_Toc24390892)

[Introduction 4](#_Toc24390893)

[Chapter 1 Review It an overview and suitability to CouchDB as well as implementing CouchDB 5](#_Toc24390894)

[1.1 Technologys used 5](#_Toc24390895)

[1.2 review it brief overview of it’s architecture 5](#_Toc24390896)

[Chapter 1 Review its suitability to CouchDB 6](#_Toc24390897)

[1.1 Drivers: 6](#_Toc24390898)

[1.2 CAP Theorem 6](#_Toc24390899)

[1.2.1 Consistency: 7](#_Toc24390900)

[1.2.2 Availability: 7](#_Toc24390901)

[1.2.3 Partition Tolerance: 7](#_Toc24390902)

[1.3 Self Contained Data 8](#_Toc24390903)

[1.4 Data replication 9](#_Toc24390904)

[1.5 No Locking 10](#_Toc24390905)

[1.6 B-Tree 12](#_Toc24390906)

[1.7 Crud On couchdb Using Postman 13](#_Toc24390907)

[1.7.1 Create 13](#_Toc24390908)

[1.7.2 Read 14](#_Toc24390909)

[1.7.3 Update 15](#_Toc24390910)

[1.7.4 Delete 18](#_Toc24390911)

[Chapter 2 Ektorp Driver evaluation as well as integration with a restful java API 20](#_Toc24390912)

[2.1 Deciding on a language in order to pick a driver 20](#_Toc24390913)

[2.1.1 Popularity 20](#_Toc24390914)

[2.1.2 Spring 21](#_Toc24390915)

[2.2 Evaluating the Drivers 23](#_Toc24390916)

[2.3 Evaluation Of Ektorp 23](#_Toc24390917)

[2.3.1 CouchDBInstance 24](#_Toc24390918)

[2.3.2 CouchDBConnector 24](#_Toc24390919)

[2.3.3 Easy Configuration 24](#_Toc24390920)

[2.3.4 Persistent Classes 24](#_Toc24390921)

[2.3.5 Out Of The Box CRUD Functionality 25](#_Toc24390922)

[2.3 Implementing EkTorp Into Java application 25](#_Toc24390923)

[2.3.1 Configuring Ektorp 25](#_Toc24390924)

[2.3.2 Persistent Entity Class 27](#_Toc24390925)

[2.3.3 The Repository/DAO Layer 32](#_Toc24390926)

[2.4 Using CRUD Functionality with Ektorp 33](#_Toc24390927)

[2.4.1 Create with Ektorp 34](#_Toc24390928)

[2.4.2 Read All with Ektorp 35](#_Toc24390929)

[2.4.3 Read 1 with Ektorp 36](#_Toc24390930)

[2.4.4 Update with Ektorp 38](#_Toc24390931)

[2.4.5 Creating embedded objects with Ektorp 40](#_Toc24390932)

[2.4.6 Updating embedded objects with Ektorp 42](#_Toc24390933)

[2.4.7 Deleting embedded objects with Ektorp 44](#_Toc24390934)

[2.4.7 Delete with Ektorp 46](#_Toc24390935)

[2.4.8 Map Reduce 48](#_Toc24390936)

[Chapter 3 React Review it’s front end 53](#_Toc24390937)

[3.1 Introduction To React 53](#_Toc24390938)

[3.2 Review It Reacts favourite page 54](#_Toc24390939)

[3.2 Add Book React functionality(Create) 56](#_Toc24390940)

[3.3 Display Books React functionality(Read) 60](#_Toc24390941)

[3.4 Upate Book React functionality(Update) 62](#_Toc24390942)

[3.5 Add Review React functionality (Update, Read 1) 66](#_Toc24390943)

[3.6 Update Review React functionality(Update) 72](#_Toc24390944)

[3.7 Delete Review React functionality(Update) 75](#_Toc24390945)

[3.8 DELETE Book React functionality(Delete) 77](#_Toc24390946)

[3.9 Number Of reviews React functionality(Map Reduce) 78](#_Toc24390947)

[Chapter 4 Data Replication and Conclusion 80](#_Toc24390948)

[4.1 Data Replication 80](#_Toc24390949)

[4.1.1 Continuous Replication 80](#_Toc24390950)

[4.1.2 Once Off Replication 83](#_Toc24390951)

[4.2 Conclusion 85](#_Toc24390952)

[References 87](#_Toc24390953)

# Title

Review It

# Summary

This paper will focus on the creation of a book review web application using CouchDB as its DataSource. It will show how to provide CRUD functionality with CouchDB via Postman. It will also show how to configure and utilize one of CouchDb’s drivers (Ektorp). The paper will demonstrate how to layer a java application as well as how to create your own RESTful API. It will demonstrate how to create endpoints and interact with these endpoints through a React front end and Postman. A choice of driver to interact with CouchDb will be discussed in detail and said driver will receive a thorough evaluation. Furthermore, the paper will discuss in depth the advantages of using CouchDb as a DataSource, it’s features and its suitability to the Review It web application.

# Introduction

For the purposes of this assignment a book review web application called Review it was created. The idea behind this web app was an online review site were users could add books to an online repository and review them. Apart from this the user would be able to see their reviews on books as well as other users book reviews. In this way the user could make an informed decision before deciding what book next to purchase. Also, users are able to keep a list of their favorite books and view all the reviews available for these books in one convenient place. In this way a user would be able to keep up to date on reviews for books they may be interested in purchasing or books that they have a genuine interest in.

# Chapter 1 Review It an overview and suitability to CouchDB as well as implementing CouchDB

## 1.1 Technologys used

* CouchDB
* Maven
* Spring Framework
* Rest API
* React.js
* Ektorp

## 1.2 review it brief overview of it’s architecture

Review It is designed to be a fully functional java web application. The system can be divided into 3 main parts. The backend was designed using CouchDb something that will be discussed in length in chapter 2 of this paper. In the middle we have a java web application. Here is where the logic of what will be displayed to the end user is manipulated and called. Maven is being used as a dependency management and build tool. To access to all external libraries such as ektorp, the developer only need to specify the dependency in his POM(Project Object Model) and he will have access to all the associated classes. Spring is being used as the java framework. Spring was developed in 2003 by Rod Johnson and since then has become the most popular application development framework for enterprise Java. Spring has many benefits too many to mention here but in the context of developing the ReviewIt web application its greatest benefit was perhaps dependency injection. Dependency injection can be used to inject a dependency of 1 class into another yet keep them separate. In this way Review it is able to have multiple layers such as a DAO layer for interacting with the database and a controller for specifying routes to deliver data to the front end.

The java application uses REST(**RE**presentational **S**tate **T**ransfer.) technology in order to deliver data to and from the front end and CouchDB DataSource. It accomplishes this by using 2 steps. First it uses a URL as an identifier this is known as an endpoint. Secondly it specifies what operation you want the server to perform on this resource using a HTTP Method. (Avraham, 2017) Usually these HTTP methods take the form of CRUD(Create, Read , Update, Delete) functionality.

Finally, the front-end was developed using the React.js. React.js is a popular javascript library that is used and maintained by facebook. It is discussed in further depth in chapter 3.

# Chapter 1 Review its suitability to CouchDB

There are several reasons why a review web app would be suited to using CouchDb as a DataSource. Below are listed some of the more prominent reasons

## 1.1 Drivers:

CouchDb contains a wide range of varying drivers that support many different languages. Since a review application would be an application that would be typically hosted on the web. This allows the developers freedom of choice when it comes to deciding which languages/platforms to develop said application in. According to db-engines.com CouchDb supports 17 different programming languages and each language has multiple drivers associated with it. The benefit of having this wide range of drivers is that it allows the developers to develop a web application using languages that they are comfortable with. (db-engines, 2019)

## 1.2 CAP Theorem

The Cap theorem states that a distributed database system can only have 2 out of 3 of the following properties. Each letter of cap represents one of these properties.

### 1.2.1 Consistency:

This means that every request receives the most recent write or a suitable error message. In this way a distributed database system is considered to be consistent because the most up to date data is being returned to the user. “simply means that each server returns the right response to each request” (Gilbert & Lynch, 2012)

### 1.2.2 Availability:

This means that data is always available for the user to be read. A request by a user will have a response. However, this response will not always be the most up to date data. Availability only concerns itself with that a request will have a response. “that the service guarantee availability. Availability simply means that each request eventually receives a response” (Gilbert & Lynch, 2012)

### 1.2.3 Partition Tolerance:

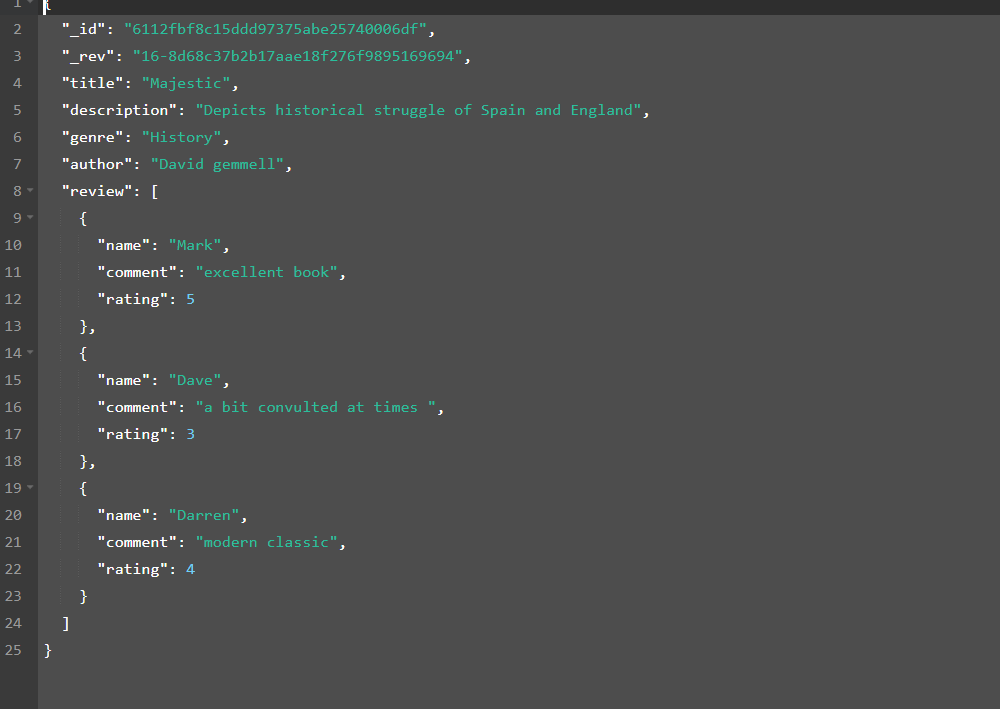
This means that systems will continue to work, despite network partitions. Messages can be dropped but aslong as the system is functioning correctly it is said to be partition tolerant.

As has been proven by this theorem a distributed database system can only belong to 2 of these 3 properties. CouchDB falls into the category of availability and partition tolerance. The reason that these 2 properties are more important than consistency for the Review it web app. Is that the user cares more about getting a response on reviews for a particular book then getting the latest and up to date review. In this way a user can make an informed decision in regards to a purchase then if he wasn’t able to retrieve any data because of consistency issues. In the choice of whether data is available or consistent. The winner for a review website would clearly be availability as the user would quickly move on to a different website if the reviews for which he visited said site for were not readily available. For this reason, CouchDB fits the application perfectly.

## 1.3 Self Contained Data

CouchDB is classified as a document store schema free datasource (db-engines, 2019). These documents are central to CouchDB’s data structure. CouchDB doesn’t enforce schemas like a relational database would. Instead what is real world data is mimicked by the CouchDB documents. Taking Review it as an example if you were to browse the favorites page which is the primary focus of the Review it web app, you would notice that a book contains none or many review’s and a review has 1 user i.e the person who left the review. In a relational database this would be represented by 3 separate entity’s, that would contain references to each other. In this way the development process would become more convoluted for the developer as now he would have to reference these 3 Tables in his application. This is only a small example, as the Review it web app is not of yet in a condition for deployment. As functionality grew this problem for developers would become more and more apparent. CouchDB bypasses this headache for the developer by using a documents store structure in were each individual document would contain a ‘view’ that would be displayed to the user.

CouchDB’ s schema less design makes it much easier for a developer to interact and create web applications such as the Review it website that I have used. Having your data contained in documents without have to worry about repeated data or a large set of tables that would add to complexity as well as development time. CouchDB allows for easy interaction with this data as it is all self-contained within the document. To illustrate this point were as a relational Database would have Book, Review, and User tables the CouchDB document contains all this information in a singular document. An example of this is shown in figure 1



Figure

Here book is represented by title to author. Review is an embedded object inside of Book and name is used to represent the user who created the review. The reason why the review web application fits this document model so perfectly is because its schema less design allows for a smoother development phase that would be greatly appreciated by the developers.

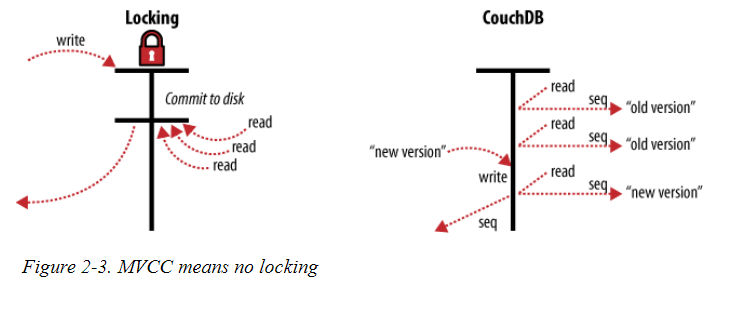
## 1.4 Data replication

The Review It application was designed as an application that would be available on a wide range of devices from pc to mobile phone and this is where CouchDB really shines. CouchDB prides itself on making it easy for the developer to set up data replication between devices and systems. The process is very simple and is discussed in further depth in chapter 3. This means that the Review It application can be replicated across a multitude of devices. Couch Db allows for two forms of replication. The first being a once off replication were the replication is performed a single time and the second being continuous replication were the replication will continue to listen to updates to the datasource. This replication method was created with error prevention in mind. It only notifies the user of a problem when user interaction is required it also is incremental in nature meaning if something was to happen before the replication had been completed it would just pick up from where it finished next time you ran the replication. It is also possible to filter your replication so that you only replicate what you need as opposed to the entirety of the data .MC Brown had this to say about CouchDB’s data replication “ only tells the user when there is a problem that requires user interaction. The replication process is also incremental, so that if anything goes wrong, such as a network outage, replication will pick right back up where it stopped. One other ability is that replication information can also be filtered.” (Brown, 2012)

The reason this is so beneficial to the Review It application is that you could replicate the data to your phone from your desktop. Filter the replication so that it would only show your favorite list. Your favorite list would be all the books that you marked as favorite and the associated reviews for each book. Then read these reviews from your favorites list while being offline. CouchDB makes this process easy and the advantage of the user not needing to be online is huge and really fits with the Review It application. This point about how beneficial CouchDB is, is summed up by Paul Krill who wrote “ if a business uses CouchDB as its server technology and a server or the Internet goes down, the business can keep running” (Krill, 2009).

## 1.5 No Locking

In a relational database if the user wanted to update a row then no one could read from that row till this update had been completed. This process is commonly referred to as a lock. “the first client gets the lock, making everybody else wait. Under high load, a relational database can spend more time figuring out who is allowed to do what, and in which order, than it does doing any actual work.” (Slater, et al., 2010) This issue with locking contributes to issues with data retrieval and can effectively slow down the entire system. CouchDB has no locking instead it uses a procedure called MVCC(Multi-Version Concurrency Control ) to manage concurrent transactions to the database. This allows for much higher speeds than a relational database. The advantage of this to the Review It application is that even as book reviews are being updated the old copy is still there to be read and when the process of updating has been completed the next read will point now to the update document. There is no enforced delay to the user and since the data isn’t critical the fact that a user read an older version isn’t adversely affecting the user. Figure 2 highlights how the MVCC procedure works.



Figure

## 1.6 B-Tree

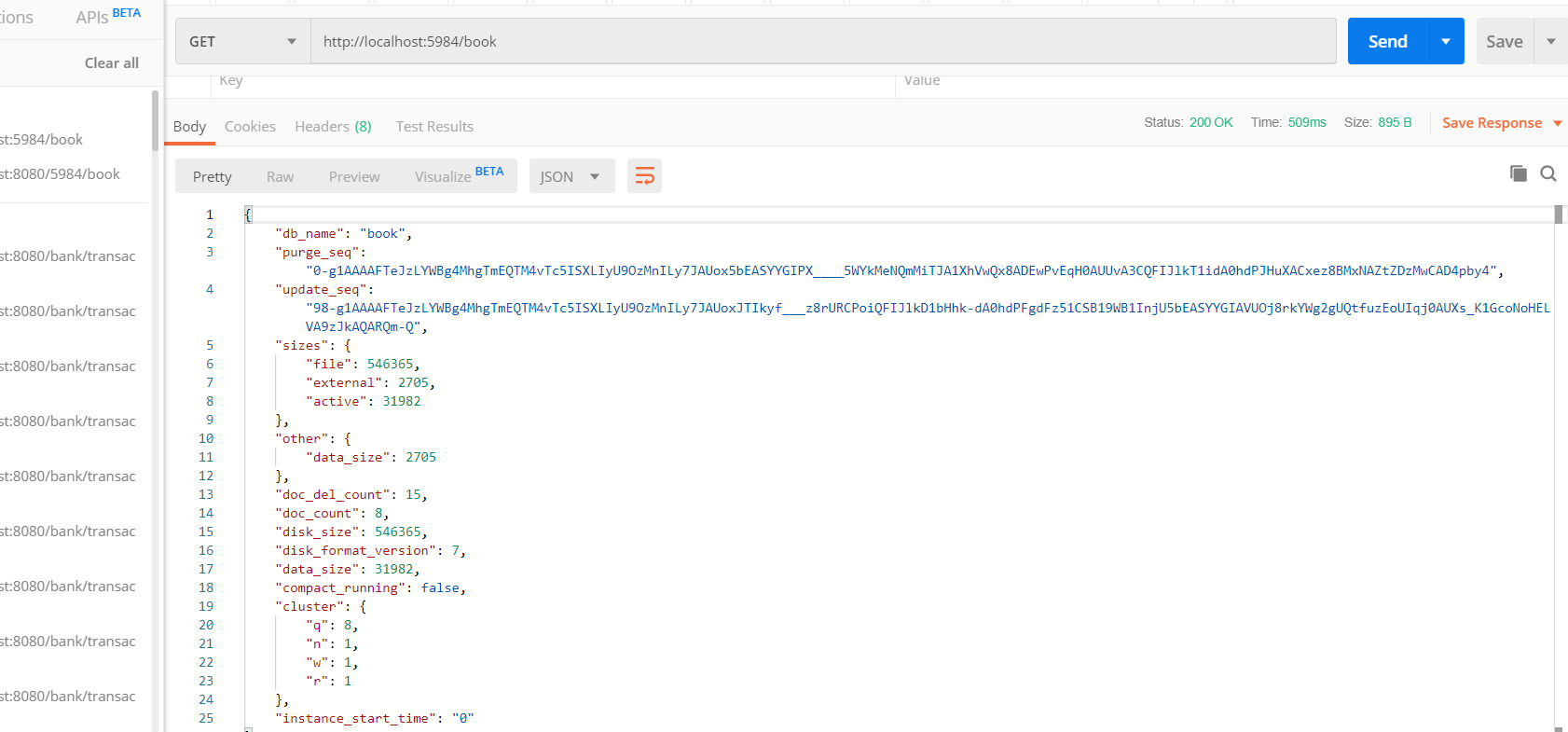
CouchDB at its core uses a powerful B-tree storage engine. B-tree is a sorted data structure that allows for insertions deletions and searches in logarithmic time. At its heart CouchDB uses MapReduce to compute the results of its views and as MapReduce produces key value pairs. Couch is able to insert them into the B-tree storage engine, sorted by the key. B-tree is the able to search the data by these keys and this leads to massive performance and speed gains when performing crud operations. The big 0 notation for these searches are *O(log N)* and *O(log N + K)*. Couch DB the definitive guide attributes the B-tree to “Being able to access results by key alone is a very important restriction because it allows us to make huge performance gains. As well as the massive speed improvements” (Slater, et al., 2010). The importance of the B-tree to the Review It application is that it improves the overall performance and query results of the entire application. Books can contain 100’s of reviews. B-tree ability to increase the efficiency to which the developer can query these objects by sorting them by keys is a massive plus to Review It as the developer is already aware that large datasets will need to be queried before a single line of code has been written.

## 1.7 Crud On couchdb Using Postman

CouchDB uses a Rest API in order to access data. Predefined routes can be used via a platform like curl or postman to perform CRUD functionality as well as view information on the database. To view information on your database the command

<http://localhost:5984/book>

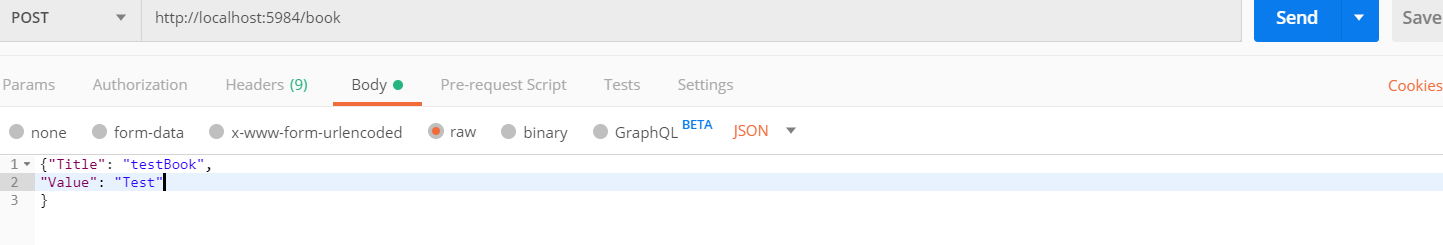
5984 refers to the port that CouchDB is listening on and book refers to the database that was created. This is a very useful command for seeing configuration information as well as general data such as number of documents, file sizes etc.



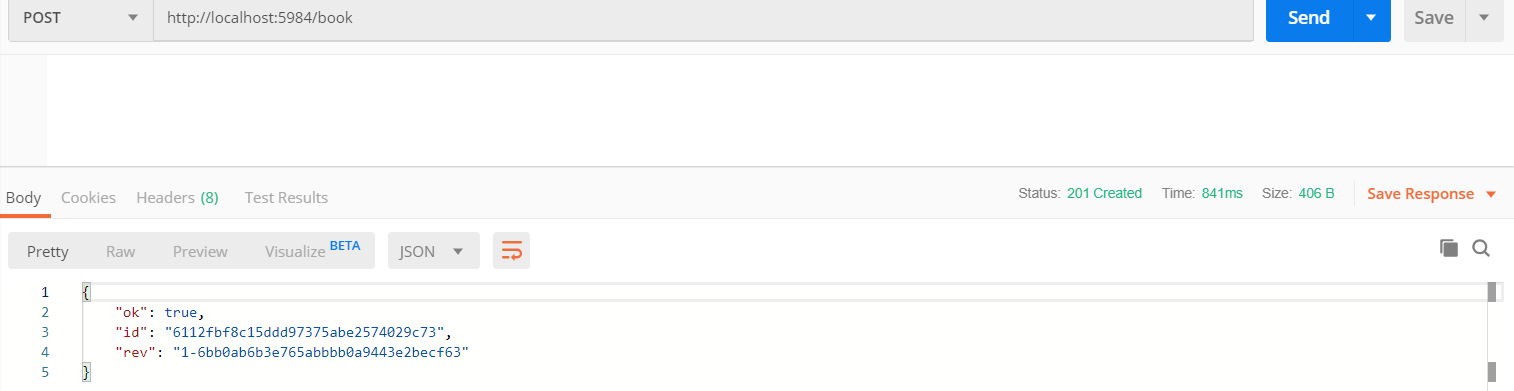
Figure

### 1.7.1 Create

The process of creating a document is easy when using CouchDB. The same URL can be used but instead of getting, a POST request will be used instead. Figure 4 shows an example of how to do this and the data that will be stored.

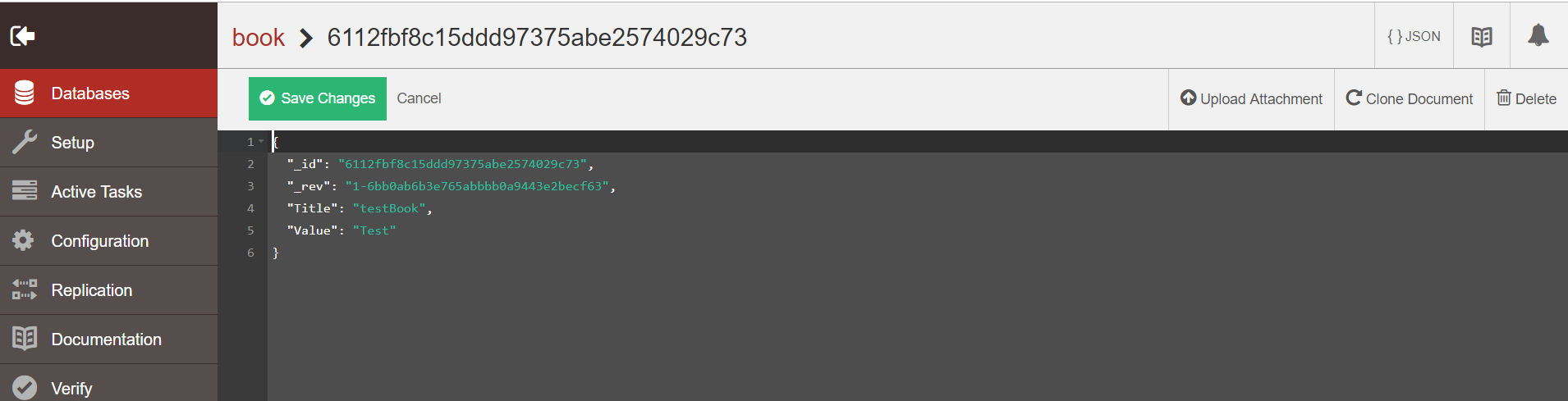


Figure



Figure

Figure 5 show the response from the post request as well as the response code 201 which stands for created. As well as that the newly created id and revision number are included in the response.



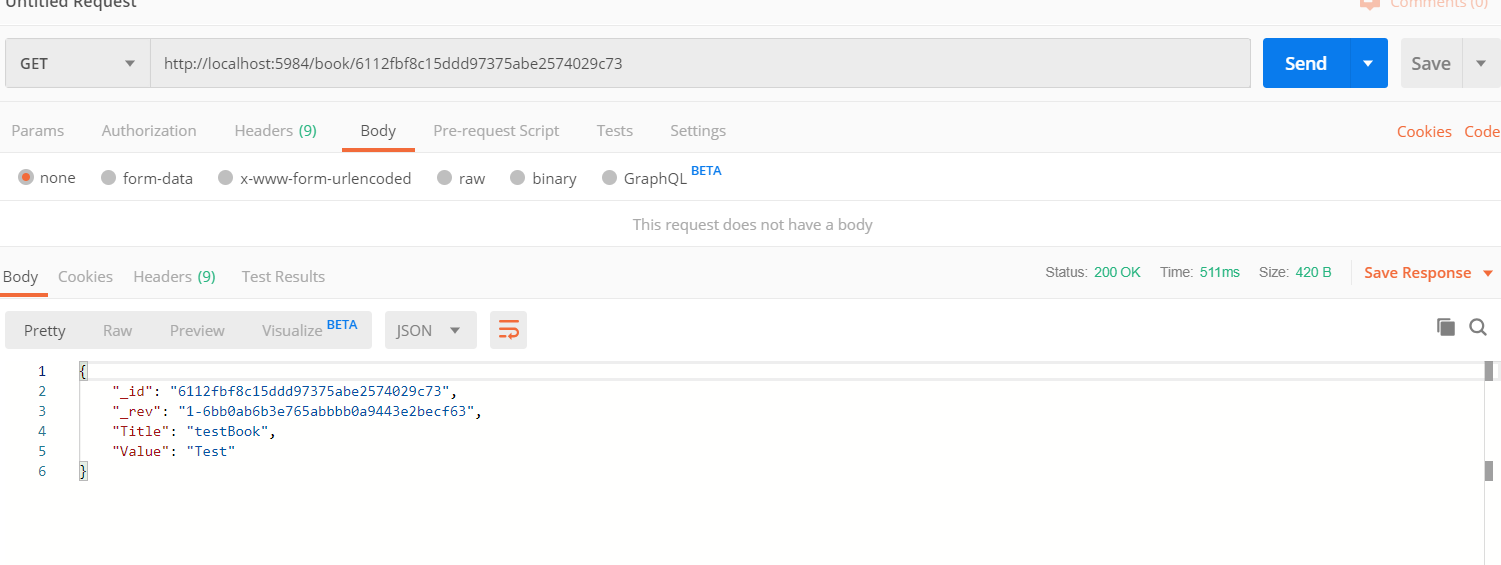
Figure

Figure 6 shows the newly created document in CouchDB. The document contains the same id and rev that the response showed in figure 5 as well as the Title and Value Information that was provided in figure 4.

### 1.7.2 Read

Now the newly created document can be read and displayed to the user. To accomplish this a new parameter needs to be passed into the request that being the id. As default CouchDB automatically creates a unique id so that it can be used as an identifier so that individual documents can be found. However, the developer can change this if he so wished. The Request will be GET request because in this case the goal is to simply fetch the data that we created in the create section of the report. The same request is used with the document id appended to the end of the request.

<http://localhost:5984/book/6112fbf8c15ddd97375abe2574029c73>



Figure

As can be seen the request returned a status code of 200 which stands for ok and shows that the request was successful. The response also shows all the data from the document in JSON format. This returned data matches the id and rev number that were created in the POST request as well as the Title and Value fields of which I added manually during that same POST request.

### 1.7.3 Update

To accomplish an update of the above document a PUT request is needed. PUT requests are used to update pre-existing documents. This request takes an additional parameter besides for the id, this being the rev number. The reason being is to ensure that the developer is updating the most recent revision of the document as opposed to an older one. The revision number in CouchDB is a unique key that changes every time an update is made to a document. It usually begins with version number but contains a multitude of other characters to make it unique. On creation it would begin with 1, after first update begin with 2 and so on.

To update the above document, we use this PUT request

http://localhost:5984/book/6112fbf8c15ddd97375abe2574029c73/?rev=1-6bb0ab6b3e765abbbb0a9443e2becf63

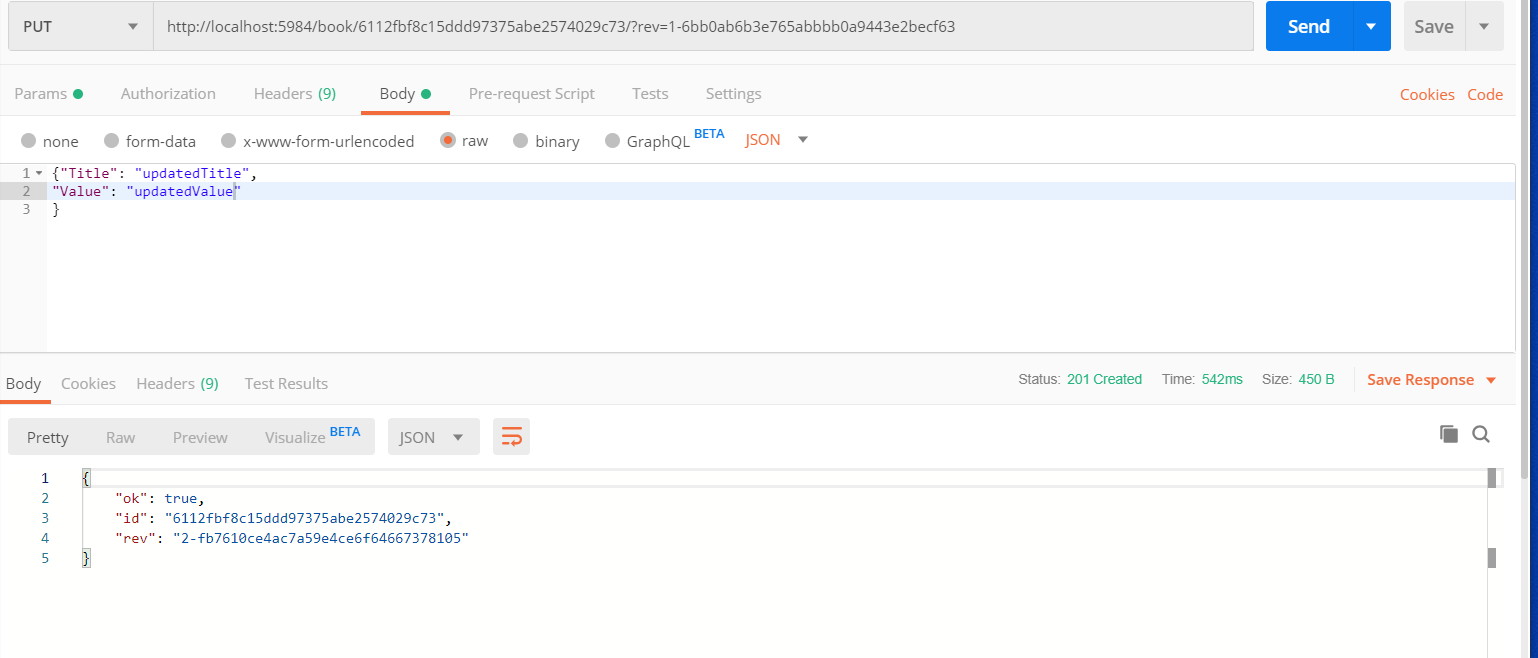
This request along with the following params

{“Title”: ”updatedTitle”,

“Value” : “updatedValue”

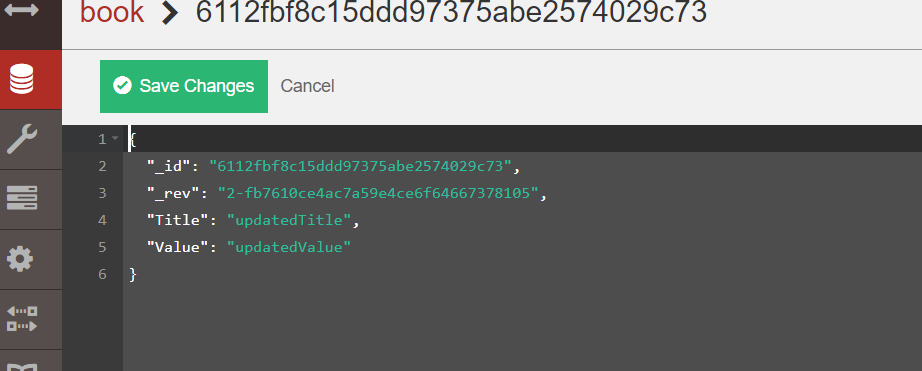
}

written in JSON will update the existing document. These values are passed into the body of the request.



Figure

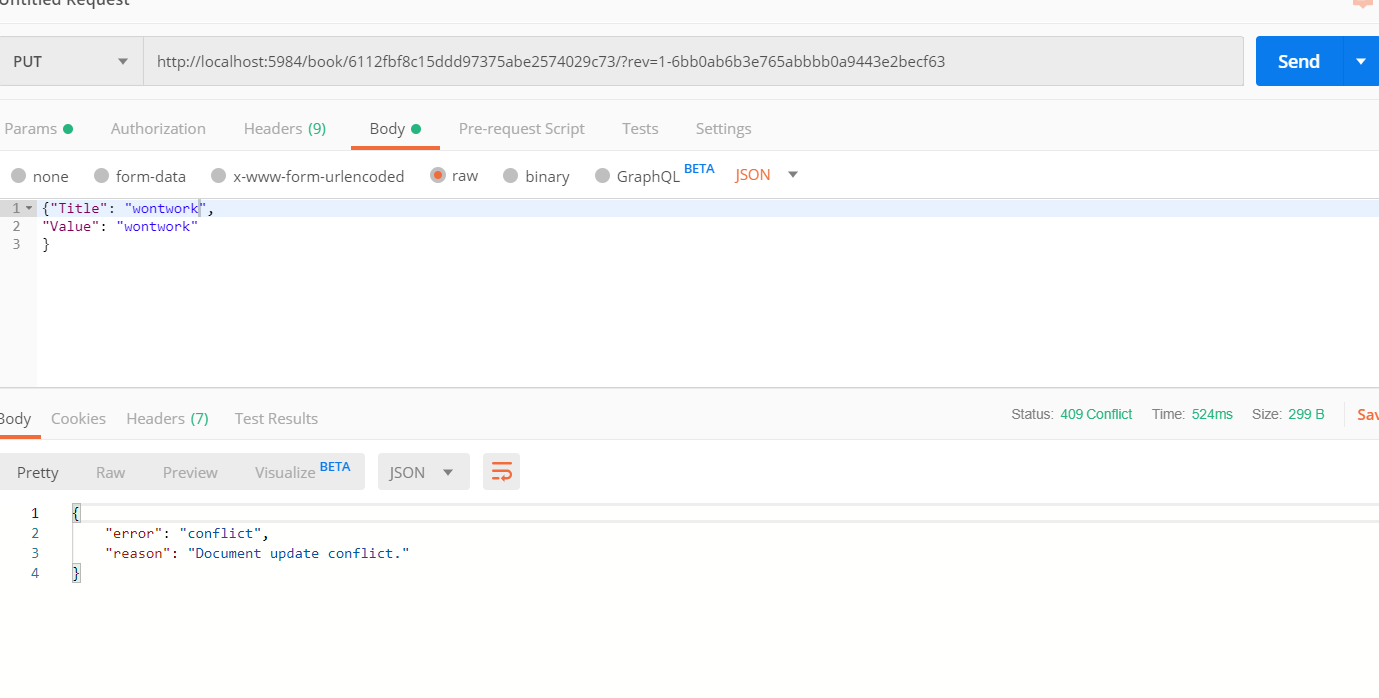
The request in figure 8 should update the title and value to the new values that are being passed in. In the bottom part of figure 8 the request received a response of 201 and a new rev value that begins with 2 i.e 2nd version of the document.



Figure

Figure 9 shows that the update was indeed successful, and the document contains the same rev value that was received in the response of figure 8 as well as the updated values for Title and Value which are also present in figure 8.

To demonstrate the importance of the rev value if I try to update the same document using the old rev number this will cause a conflict and not allow the update to occur.



Figure

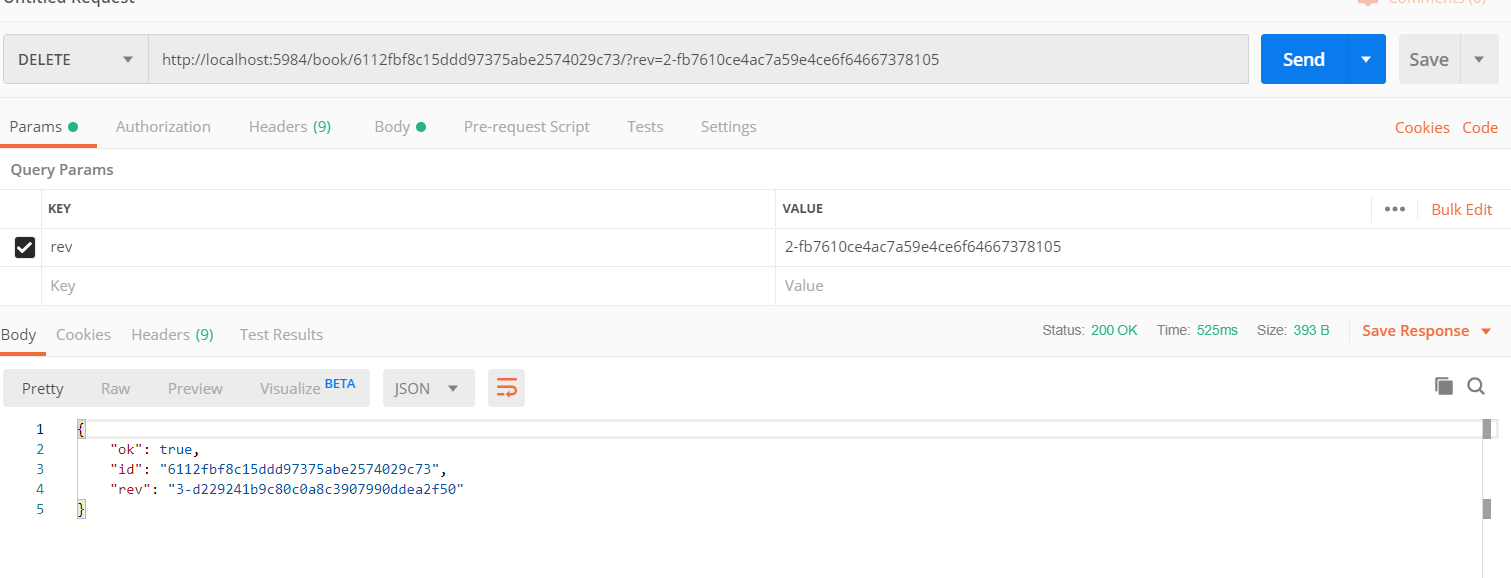
As can be clearly seen, CouchDB doesn’t allow the update. A status code of 409 which means conflict is returned as well as an error and reason fields. CouchDb specifies the reason for the error is Document update conflict. This clearly demonstrates the importance of the rev number and how CouchDB deals with updates of older documents.

### 1.7.4 Delete

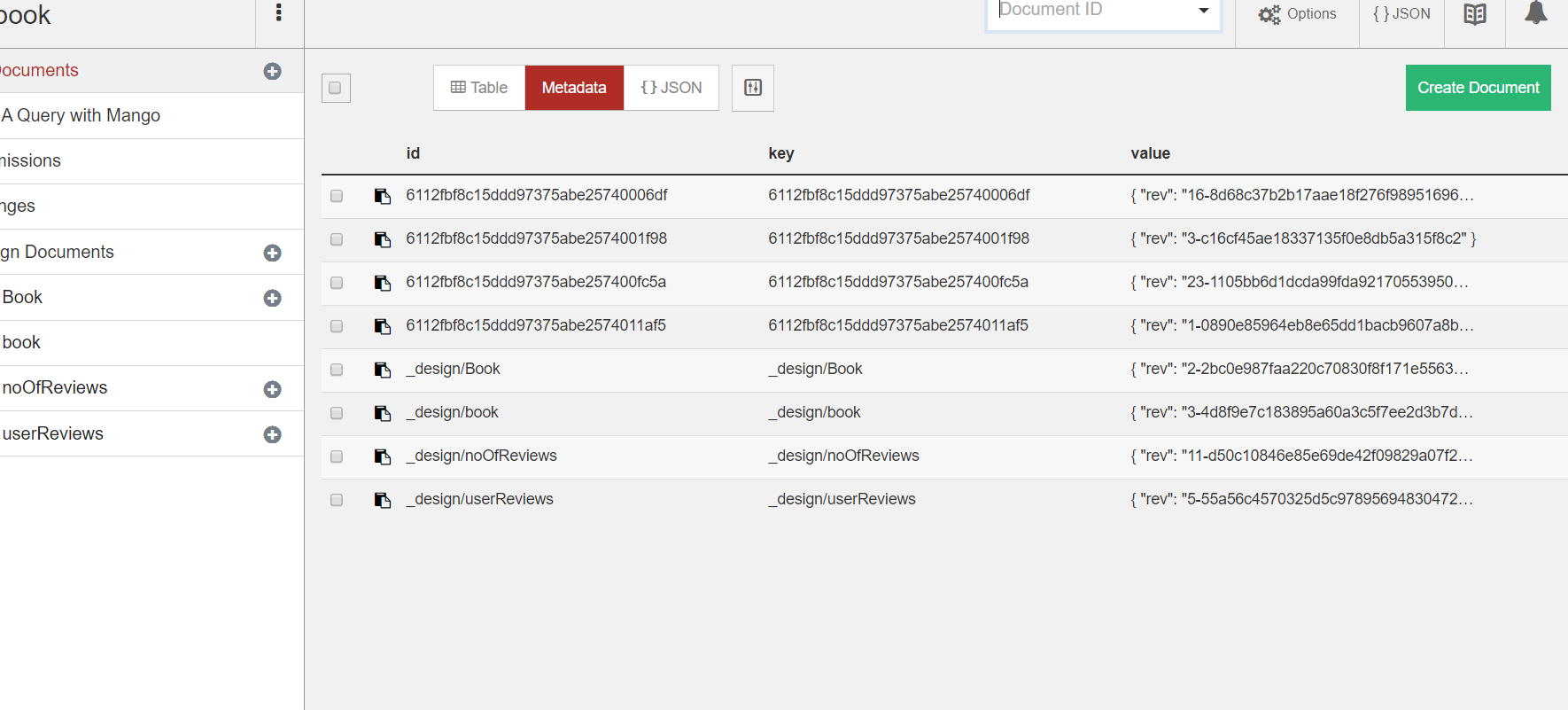
To delete a document from CouchDB a delete request is needed. Much Like the PUT request a delete request needs an id and a rev value. The reason why it needs a revision no is because of CouchDB’s nature in where a document is never truly deleted but only marked as such. A field marked \_deleted is instead set to true to let CouchDB know not to display this document anymore.

To delete the document the following delete request is made

<http://localhost:5984/book/6112fbf8c15ddd97375abe2574029c73/?rev=2-fb7610ce4ac7a59e4ce6f64667378105>



The delete request returns a status of 200 OK meaning the delete was successful. However, it also sends back an id and a new rev value. The importance of this new rev value is that it proves that the document was never in fact deleted rather couch has marked it as deleted instead. The rev value beginning with 3 indicates that this is the 3rd change that has been made to that document. To prove that the document has been marked as deleted a look at CouchDB now shows us that there is no document with id 6112fbf8c15ddd97375abe2574029c73 available to the user. The contents of CouchDb can be viewed in figure 11.



Figure

# Chapter 2 Ektorp Driver evaluation as well as integration with a restful java API

## 2.1 Deciding on a language in order to pick a driver

Well to begin any discussion about evaluation of drivers it is most important that a language be first decided upon. CouchDB provides driver support for 17 different languages, each of these languages can contain 4 or more drivers. To narrow the driver search down a language had to be decided on first. The Review It application was designed in java for a number of reasons.

### 2.1.1 Popularity

Java is a hugely popular language in the development industry. It is instantly recognizable and has been a mainstay in list of top 5 most popular coding languages. Stackify rates java as the most popular programming language in the world as of august 2019. They say that “If you know Java, chances are you won’t be desperate for work!” (Putano, 2019)

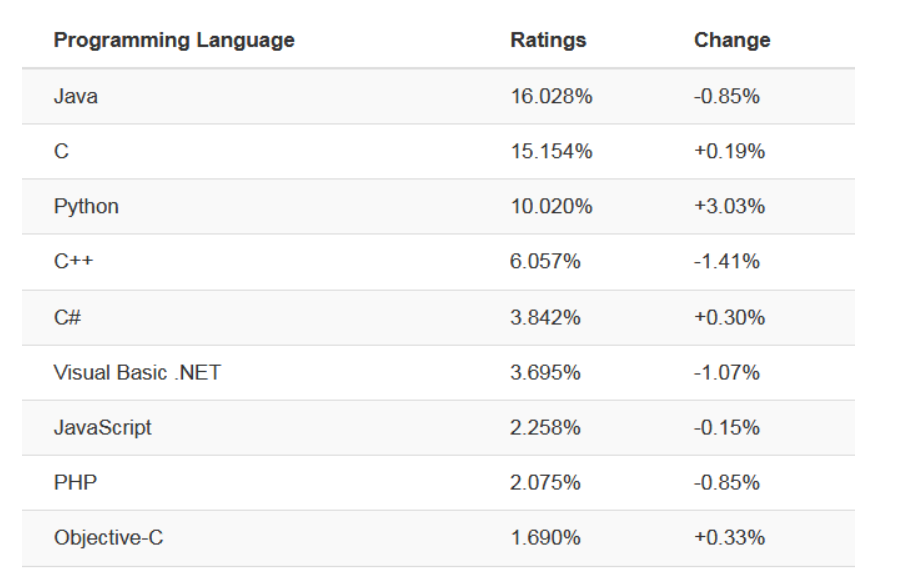
Figure 12 shows stackify’s ranking of the most popular programming languages 

Figure (Putano, 2019)

Tiobe one of the world’s leading software quality company’s produces an index that ranks programming languages by popularity. Through research a lot of sites quote this index when citing the popularity of a language. As of the 1st of nov 2019 it cited java as the most popular language. However, what is more relevant is java’s stability and clear ongoing popularity since 1999. As can be seen in figure 13 java has not dropped out of the top 3 programming languages since 1999.

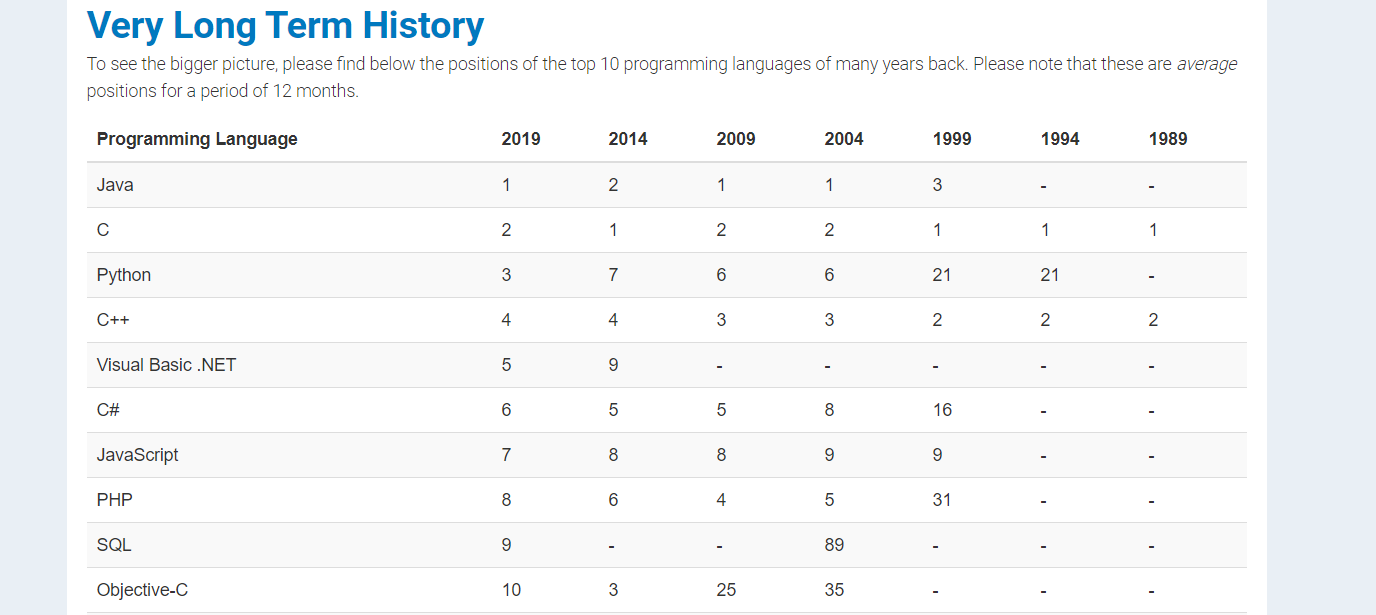


Figure (Tiobe, 2019)

### 2.1.2 Spring

The spring framework is a powerful multifunctional framework that provides massive benefits to web application development. However, Spring is only available through use of the java language. Although the benefits are too many to list here the main advantages in relation to the Review It web application are

#### 2.1.2.1 ORM(Object Relational Mapping) Mapping

ORM is a powerful tool for developers. ORM allows the developer to persist his database onto objects then manipulate these objects instead of making direct calls to the database. In this way the developer is creating a layer of abstraction between the DataSource and the web application. Spring interacts with most of the ORM tools that are available and through use of annotations support its practice. Spring also provides support with error handling and most persistence exceptions which otherwise would not be recoverable from. “This feature allows you to handle most persistence exceptions, which are non-recoverable, only in the appropriate layers, without annoying boilerplate catches, throws, and exception declarations.” (Spring, 2019) When dealing with a DataSource abstraction is generally considered the best practice.

#### 2.1.2.2 Loose Coupling

Spring through its use of dependency injection achieves loose coupling. When a web application becomes too large, objects tend to become highly coupled. The problem with having highly coupled objects in your system is that the system becomes hard to modify or alter. This is because the classes that compose the system are dependent on each other and a change to one class could break another part of the system that the developer hadn’t anticipated on. The Spring framework allows for easy managing of the dependencies between objects. How Spring achieves this loose coupling is through the use of annotations such as @Autowired or @Inject. By using these annotations, the 2 classes can have interact with each other but have very little knowledge of what the other object can do. Loose coupling has huge advantages to any application

* Better Testability:

Since your code is not highly dependent on other objects and they are just passed in as needed Unit test are easier to write. (Danylko, 2016)

* Modifiable:

Since classes aren’t dependent on each other less risk of breaking existing features when adding new code “ can write additional code without breaking existing functionality and feel safe writing it.” (Danylko, 2016)

* Swappable Components:

Most developer’s strife to produce plugin architecture. Plugin architecture is where it is easy to switch major components of your application without much additional work. Examples of this would be changing the UI from Swing to React without effecting your java application or switching DataSources “ If you want to swap out the Oracle database component with a SQL Server component, if developed properly, it can be done easily.” (Danylko, 2016)

The reason all this sits so well with the Review it application is that it is an evolving application. Spring makes it easy to add new functionality without fear of breaking existing classes. It also offers easy unit tests for the developers as well as making it so that the developer isn’t dependent on one particular component e.g React.js. As technologies and frameworks improve so to can the application.

## 2.2 Evaluating the Drivers

Having decided on a language that left a choice of 4 drivers. These being java-cloudant, Ektorp, LightCouch and CouchRepository. Immediately ruled out CouchRepository which has only 1 version on Maven Repositories and hasn’t been updated since jan 2015. Java-cloudant was dismissed because it wasn’t created with CouchDb in mind. Java cloudant was designed to be used by NoSQL databases and offers support for many differing databases. The advice would be to avoid anything but a dedicated driver when using it to interact with a DataSource. That left LightCouch and Ektorp. Both these drivers have many releases meaning the developers of said drivers have interest in improving them and eliminating bugs. The latest version for both was late 2017, Ektorp(sept 2017) and LightCouch(Nov 2017). Based on these facts not much separated the 2 drivers however when it comes to online documentation or community advice Ektorp definitely has an advantage over LightCouch. This was the primary reason for choosing this driver.

## 2.3 Evaluation Of Ektorp

There are many benefits associated with using Ektorp. These are but not limited to

### 2.3.1 CouchDBInstance

Through use of the CouchDb instance interface, the developer is provided with methods for managing multiple databases. In this way the application can use multiples CouchDb DataSources in his application.

### 2.3.2 CouchDBConnector

Through use of the CouchDB Connector interface the developer is provided with methods for interacting with documents from a specific database.

### 2.3.3 Easy Configuration

Ektorp prides itself on minimal configuration. Ektorp well documents how to establish a connection with your database and get access to the previously mentioned CouchDB Instance and CouchDB Connector. Only a minimal amount of code is required in order to accomplish this feat as can be seen in figure 14

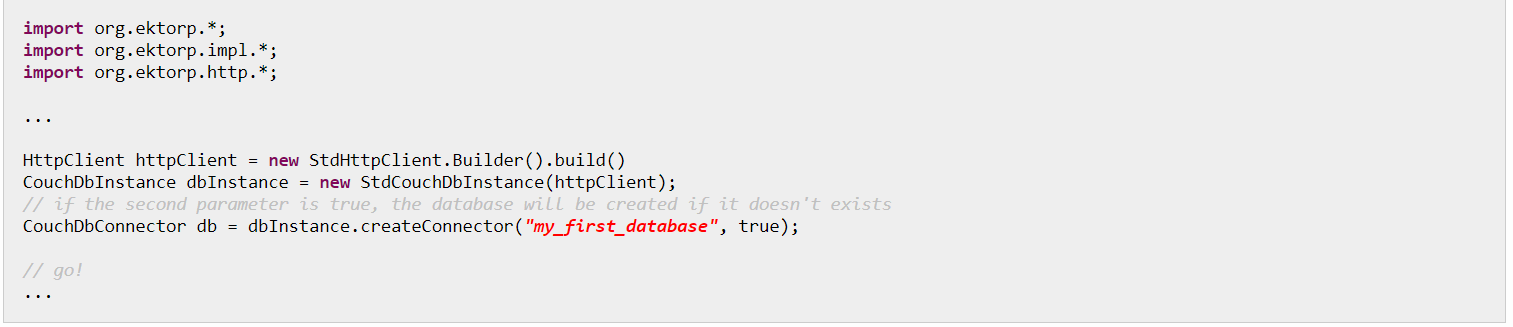


Figure (Lundgren, 2017)

### 2.3.4 Persistent Classes

One of the major benefits of Ektorp is how its use of persistent classes compliments Spring so well (See Spring ORM section above for details). Ektorp creates persistent classes in the same way that any other popular ORM tools would “Ektorp is mainly build for persisting rich domain classes in CouchDB much like classing ORM tools such as Hibernate” (Lundgren, 2017). This persistence is achieved by using powerful object mapping features provided by the Jackson library. The advantages of using an ORM tool has already been covered in section 2.1.2.1. But the fact that Ektorp allows you to use it in this same manner is a massive selling point of the driver.

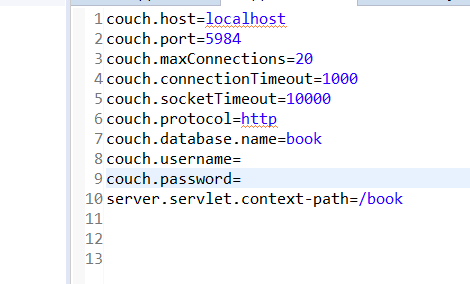
### 2.3.5 Out Of The Box CRUD Functionality

Ektorp gives you out of the box CRUD functionality through use of the CouchDbRepositorySupport class. All that is needed is to extend this class to gain access to its predefined methods. These methods include add, remove, update and various different gets. In this way Ektorp is very developer friendly and easy for a developer to get a CRUD application working.

## 2.3 Implementing EkTorp Into Java application

### 2.3.1 Configuring Ektorp

The first step involved adding in the database information to my application.properties file. Typically the application.properties file deals with various configuration information that your application may need. This file in a spring application is always located under src/main/resources. Figure 15 shows the contents of the application.properties file.



Figure

Once the configuration details have been added to the applications.properties file they can be used in the CouchDbConfig file. The CouchDbConfig file can be seen in figure 16.



Figure

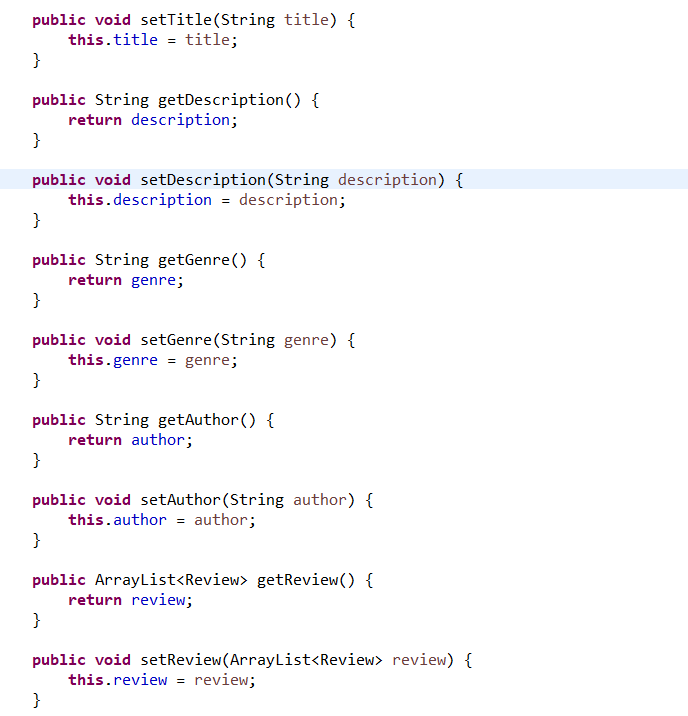
As can be seen from figure 16 the configuration is in fact easy to create. The private attributes that are being declared their values are coming directly from the application.properties file. This is accomplished through the use of annotations. The @Configuration specifies the class as a configuration class. The @Configurationproperties prefix = “couch” is specifying that the values are coming from the application.properties file. Finally, the @EnableConfigurationProperties is allowing these properties to be used by the configuration class. It also uses the CouchDbInstance and CouchDbConnector which were discussed in sections 2.3.1 and 2.3.2.

### 2.3.2 Persistent Entity Class

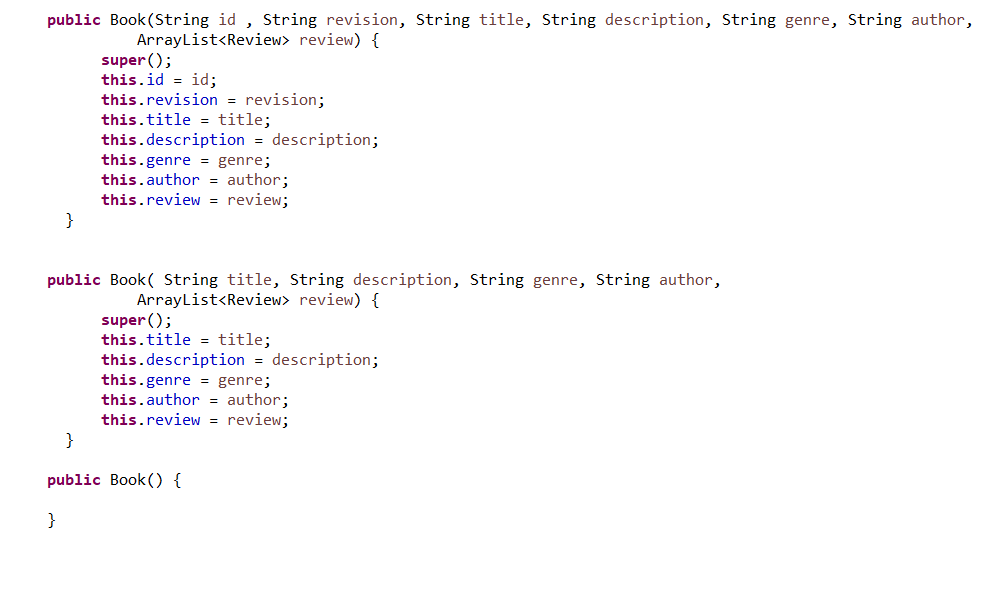
The Book Class in the Review It application is a persistent entity class in the java application. The class represents the data that will be contained within the documents of CouchDB. The Book Class contains attributes that relate to a book as well as an arrayList of Reviews. The reason it contains an arrayList of type Review is because the intention is for the book document to contain many embedded Review objects. As a book can have none or many Reviews. It is important to note that for the Ektorp to persist the data onto the entity class the id and rev fields have to be annotated with @JsonProperty. The contents of the Book class can be seen in figure 17 ,figure 18 an figure 19



Figure

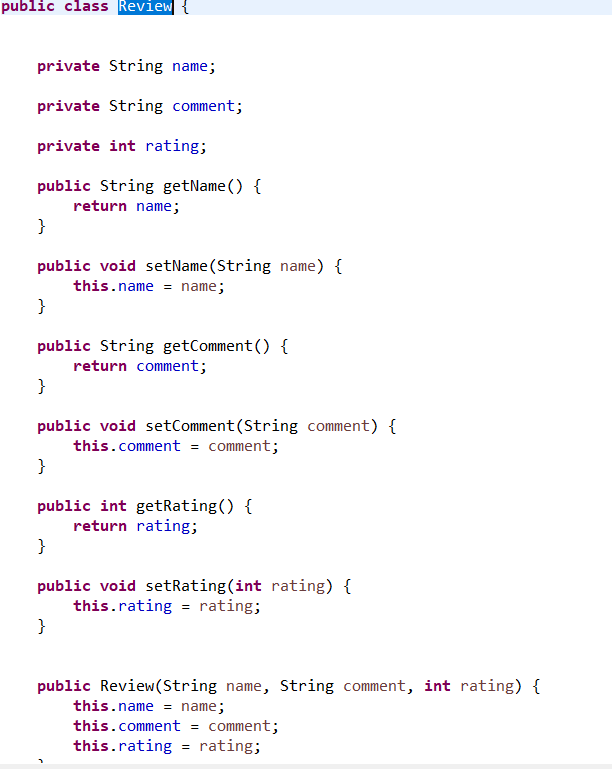


Figure



Figure

As can be seen from above the Book class will store information on id, rev, title, description, genre, and review object. The review objects contain information on name, comments and rating of the review. The class and its contents can be seen in figure 20.



Figure

### 2.3.3 The Repository/DAO Layer

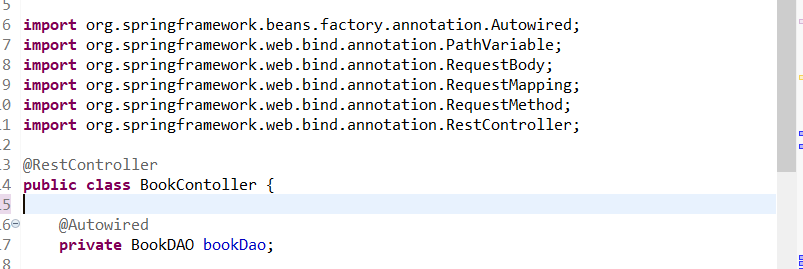
The final step of configuring Ektorp was to create a class that extends CouchDbRepositorySupport. The class that I created was the BookDAO class. By extending this class it gave BookDAO access to the full CRUD functionality of the Ektorp driver. Figure 21 shows the contents of the BookDAO class. 

Figure

As can be seen from the image the BookDAO constructor accepts the CouchDbConnector as a parameter. In this constructor the Book class is specified as a persistent class. Meaning that the CouchDbConnecter is aware that the Book Class is a representation of the documents contained within CouchDB.

## 2.4 Using CRUD Functionality with Ektorp

Now that the configuration has been completed. The process of adding CRUD functionality can begin. To accomplish this, I began with creating a controller class for managing the CRUD functionality. This is a common way to design web applications. The controller class is where the developer can specify his endpoints. Each individual endpoint performs its own specific task. These endpoints can be hit from tools like postman are entering them into a URL. Review It uses them so that the React Front End can interact with these endpoints to get relevant information and perform CRUD functionality as needed. In order for the Controller to be able to perform this functionality it needs access to the book class. In order to loosely couple these too classes together the @Autowired annotation was used. The code for creating the RESTful controller and loose coupling can be seen in figure 22.

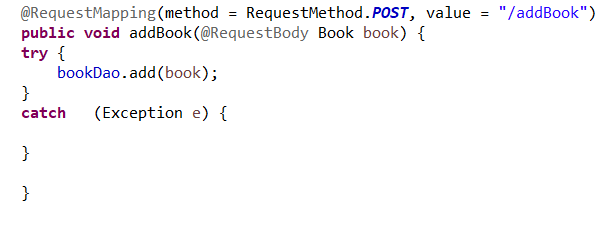


Figure

Figure 22 shows how easy Spring classes are to create. The framework itself is annotation driven and through the use of small annotations you get a lot of functionality. The above lines of code have created a REST controller through use of the @RestController annotation and loose coupling through the use of @Autowired.

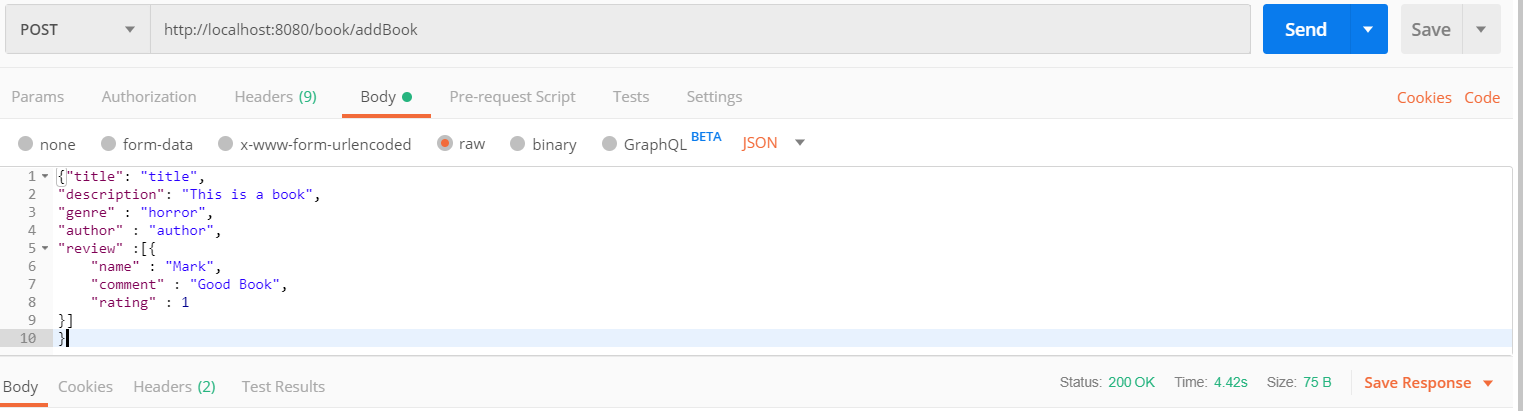
### 2.4.1 Create with Ektorp

The endpoint that was created in order to create a book was /addBook and can be seen in figure 23



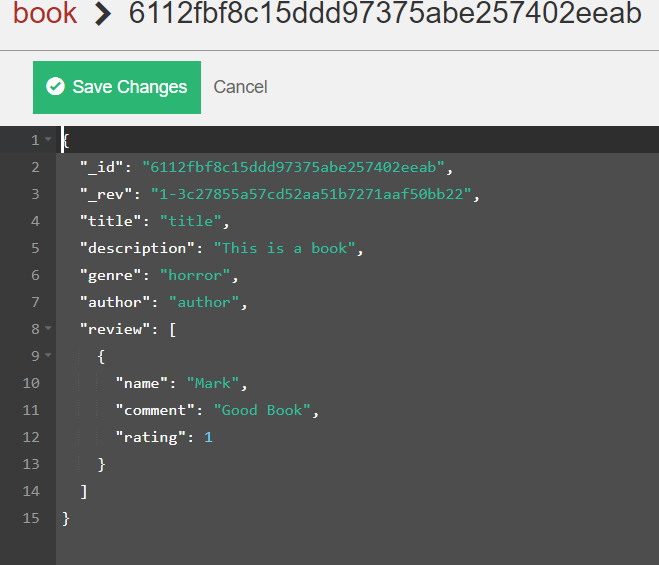
Figure

As can be seen it’s a post method that has access to bookDao dependency’s add method. The @RequestBody annotation will convert the body of the HTTP request to an object of type book. Below in figure 24 is an image of the endpoint being tested through Postman.



Figure

As can be seen a body is being added to the request. The body also contains an embedded object of type Review. This request received a response code of 200 OK which seems to suggest that the was processed correctly.

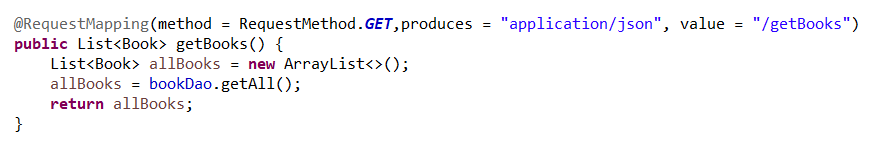


Figure

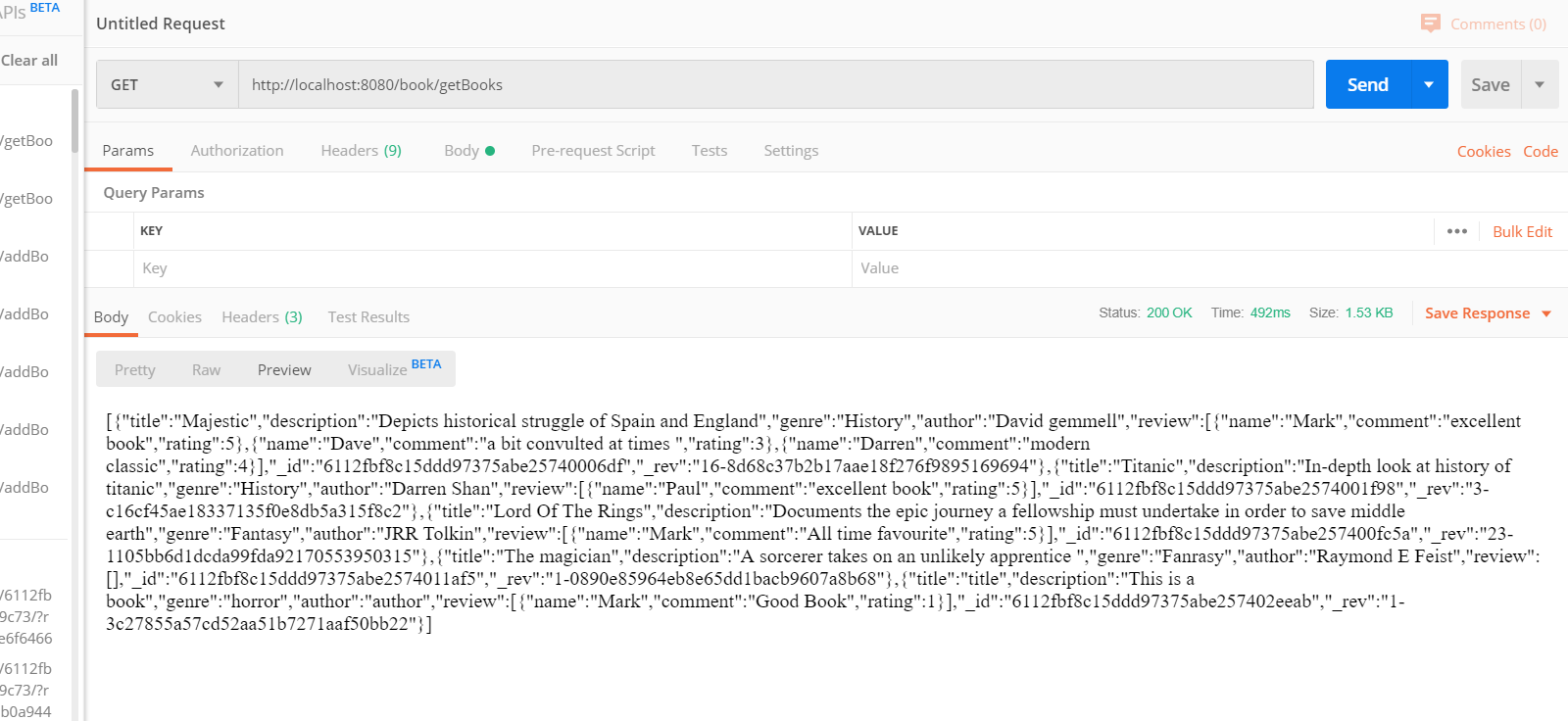
As can be seen in figure 25 the correct details were saved into CouchDB. Couch added its on id and revision values, but the rest of the data came directly from the body of the post request from figure 24.

### 2.4.2 Read All with Ektorp

In order to read all the data from the database the following endpoint was set up /getBooks.



Figure

As can be seen in figure 26 the mapping shows that it is a get method that produces JSON. I create an array of books and let the empty area equal to the getAll() method. The getAll method is a predefined method of Ektorp and acts on all the documents in CouchDB and not just any specific 1. 

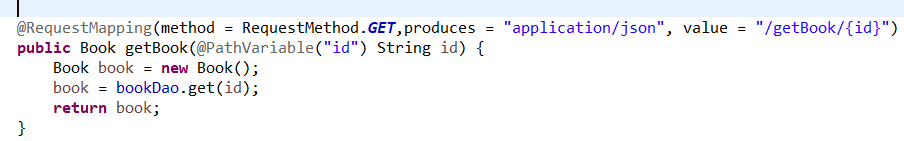
Figure

Figure 27 shows the request receiving a status code of 200OK. The more noteworthy information that figure 27 contains however is every single book that is currently inside the database. Showing that the request was successful and did in fact work.

### 2.4.3 Read 1 with Ektorp

In order to read the data of a specific document the following endpoint was set up

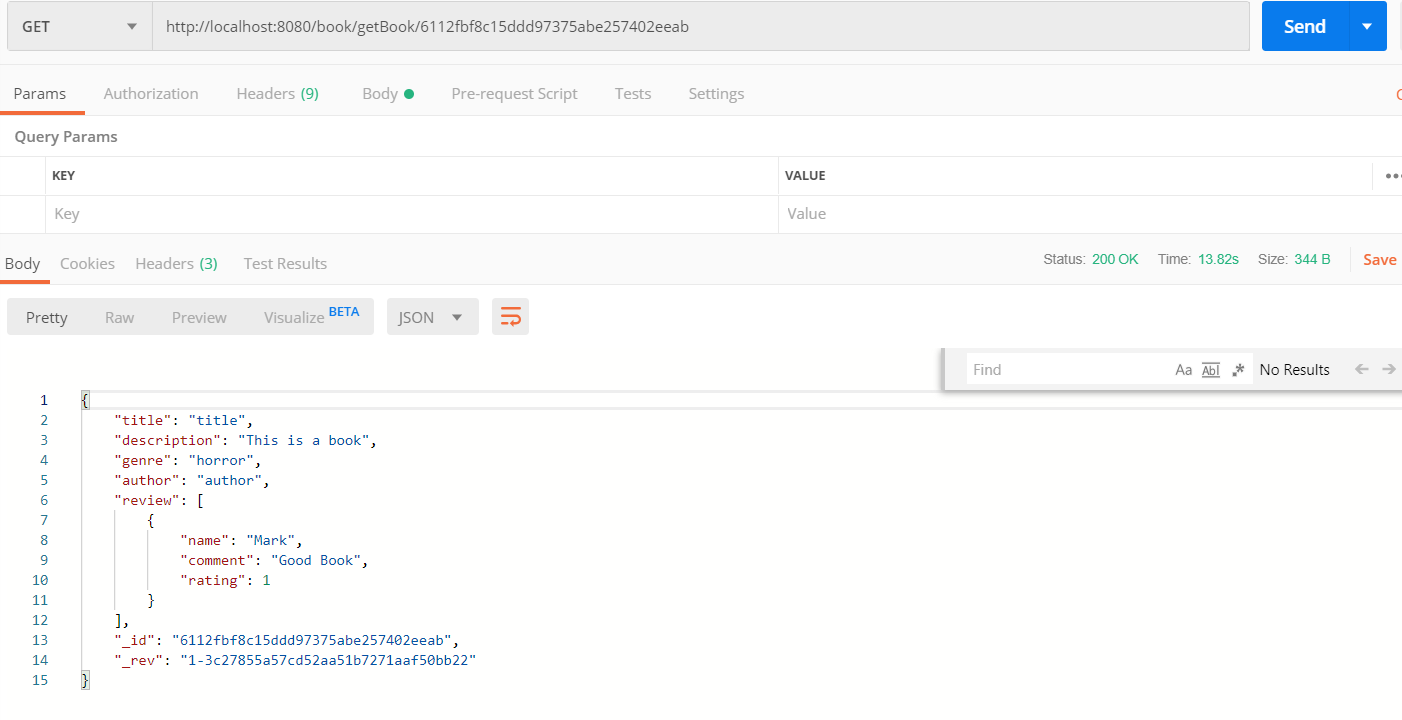
/getBook/{id}



Figure

Figure 28 show the endpoint for getting the information associated with 1 book.

The @PathVariable annotation signifies that a String id will be extracted from the endpoint, specifically the {id} part of the endpoint. Once the id has been extracted it can be used by Ektorp’s predefined get method. The results of the endpoint should be 1 specific document as the id is unique.

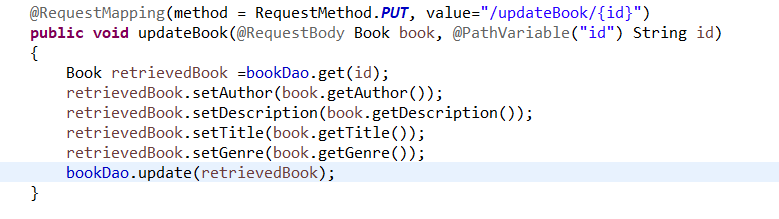


Figure

As can be seen in figure 29 the request received a response of 200 OK suggesting that the request was successful. Furthermore, a JSON body was returned by the response. The information in the response matches exactly the information used for the create in section 2.4.1.

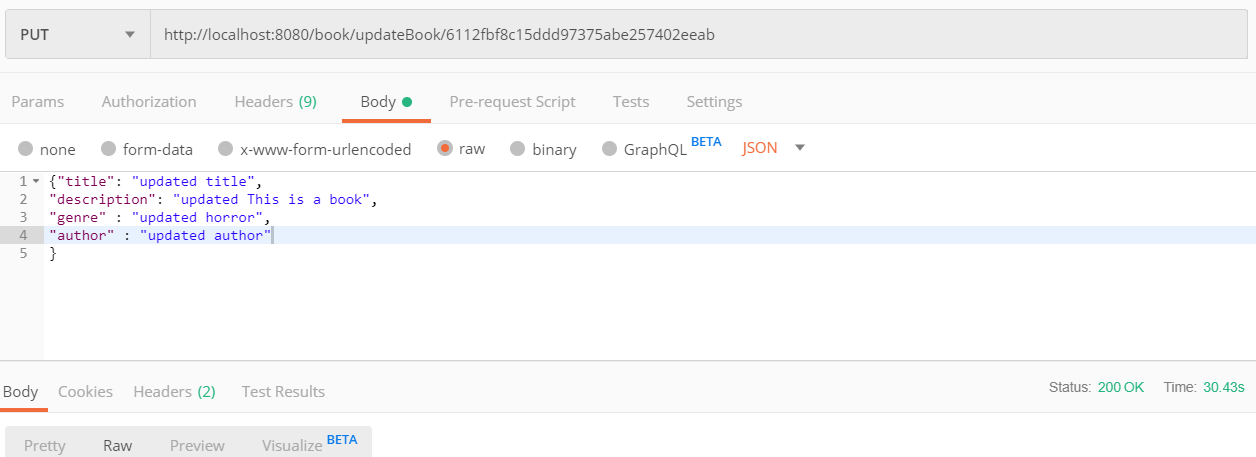
### 2.4.4 Update with Ektorp

In order to update a document in the CouchDb database an update endpoint was created. The endpoint used was /updateBook/{id}



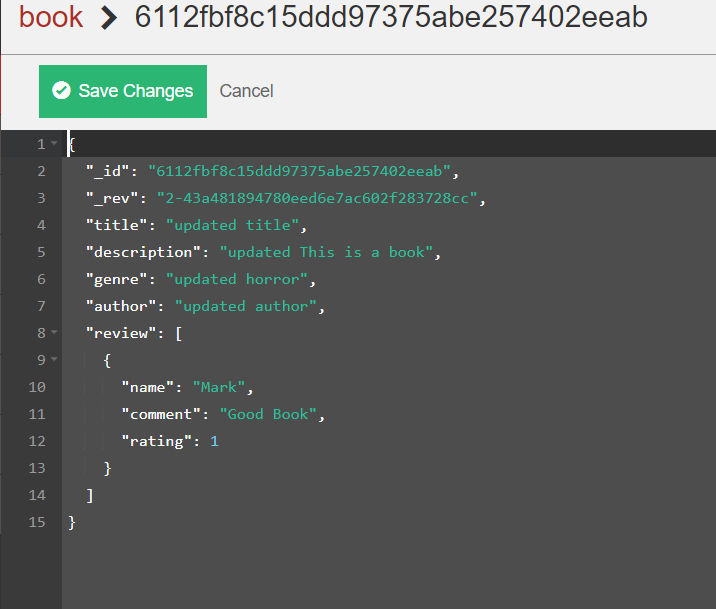
Figure

As can be from figure 30 the @PathVariable id is populated through the endpoint. While the @RequestBody book object is populated by the body of the received HTTP request. The request method of the endpoint is PUT which relates to the fact that this endpoint deals with updates. How the update was accomplished was by getting the values associated with a Book object and using setters to alter the fields. The reason it is done in this way is that this endpoint is looking to update the book object only and not the embedded review object. If the get was not included all reviews associated with the book would be wiped out on update. Finally, the predefined Ektorp update method is used to update the document.



Figure

Figure 31 shows the update endpoint being tested in postman. A JSON body of updated values to match the book object were added. Also, the request received a response code of 200 OK which suggests that the update was successful.



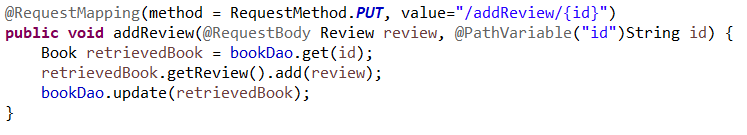
Figure

Figure 32 shows that CouchDB was indeed updated with the new values. As is evidenced by the picture the reviews remain but the book fields have been altered. The document has received a new revision number. This can be seen by comparing it to figure 25 which is the same document.

### 2.4.5 Creating embedded objects with Ektorp

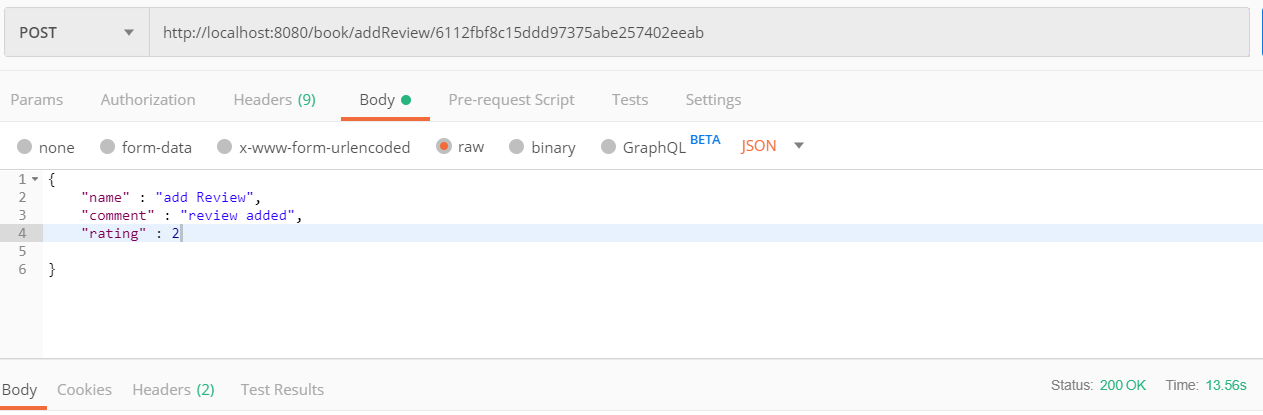
Before showing the delete in action it is important to show how the reviews, which are embedded objects in the Book document are handled. Since the Reviews make up the main part of the Review It application being able to manipulate them was crucial to the success of said application. To begin with, the process of adding a review will be looked at. The endpoint that was used to add a review is

/addReview/{id}



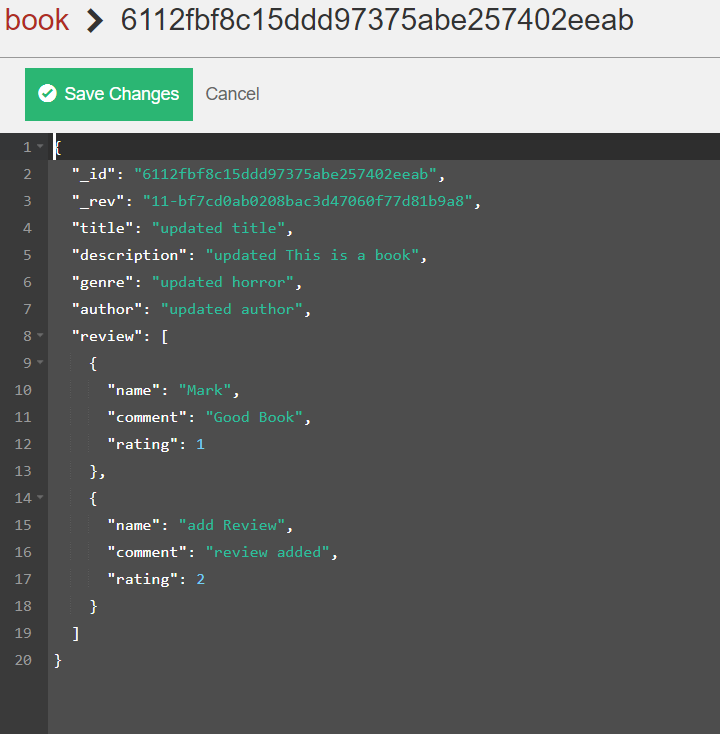
Figure

The addReview endpoint uses a PUT request method. That is because even though a review is being added we want the document to retain the pre-existing information on the book and its other reviews. To accomplish this the id is used to retrieve the document from CouchDB and populate a book object. The book objects array of reviews is then accessed, and the new review is added to that list. Finally, the updated object is updated through use of Ektorps update method. The Review object is populated through the request body and the id is got by extracting the value from the endpoint.



Figure

Figure 34 shows the request receives a status code of 200 OK which suggests that the review has been added.



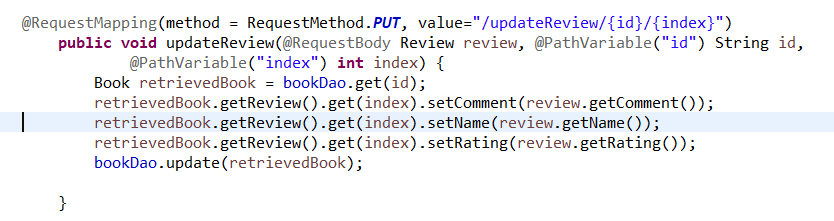
Figure

It can be seen in figure 35 that the new review was added successfully to CouchDB and that that the pre-existing book info and review info remain untouched. This show that the process of adding a new review has worked as intended.

### 2.4.6 Updating embedded objects with Ektorp

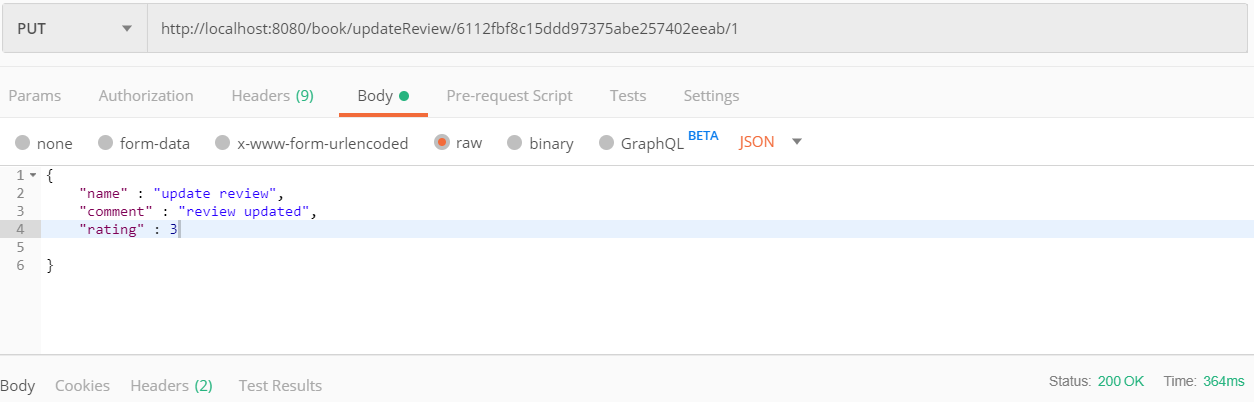
The process of updating a specified review proved trickier than creating one. To accomplish this task the following endpoint was created.

/updateReview/{id}/{index}



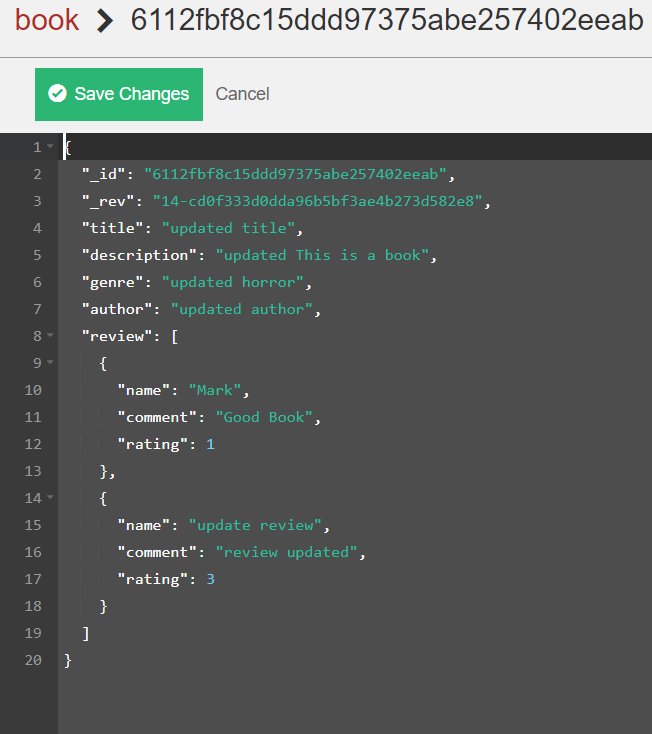
Figure

The updateReview endpoint takes a in a review object and an id just like the addReview method. However, this endpoint requires another parameter. This being the index of the array of Review objects. The index number will match the position in the array of the review that the user wishes to update. As can be seen from figure 36 the index number is used to get the review object intended for update and the setters are used to add the values received from the endpoint. Once the values have been changed the Ektorp update method is used to update the document.



Figure

Figure 37 shows the PUT request being tested through Postman. The body contains the JSON review details that are being used to update the review object. The URL contains the index of the review that was created in section 2.4.5. The request received a response code of 200 OK.



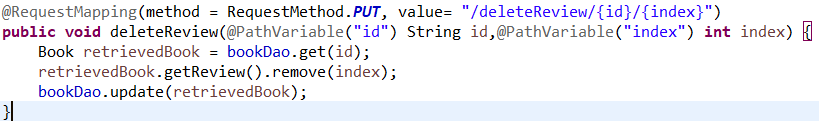
Figure

Figure 38 shows that the review has indeed been updated. As well as this the picture proves that the endpoint is working as expected. Bar the intended object and the rev number none of the other details have changed.

### 2.4.7 Deleting embedded objects with Ektorp

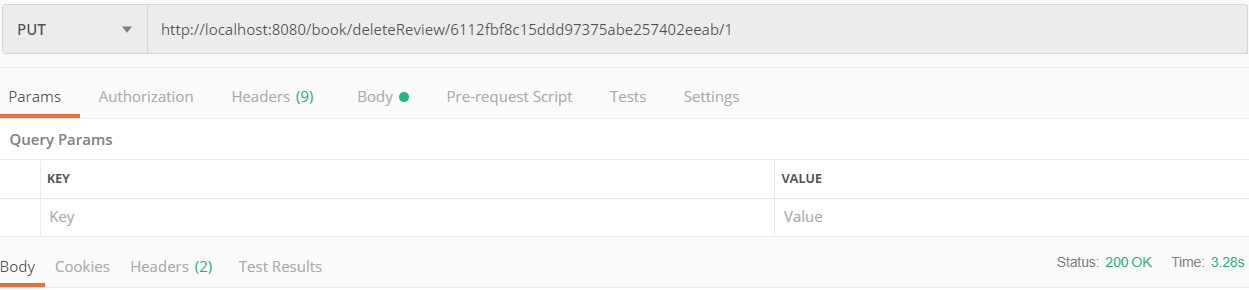
To delete a review from CouchDB the following endpoint was created.

/deleteReview/{id}/{index}



Figure

Figure 39 shows the delete review endpoint. As can be seen the endpoint takes in 2 params those being the document id and the position of the review in the review array. Ektorps get method and the id is used to populate a book object. Then the index is used to remove the intended review object from the array. Finally, the Ektorps update method is used to update CouchDB.



Figure

Figure 40 show the endpoint being tested in Postman. The URL contains the document id and the index of the array intended for deletion. On the bottom right of the image the request received a 200 OK.



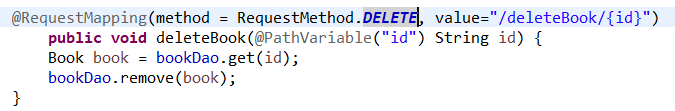
Figure

Figure 41 shows that the endpoint works as expected. The review isn’t present in the document anymore and all the other data has remained untouched.

### 2.4.7 Delete with Ektorp

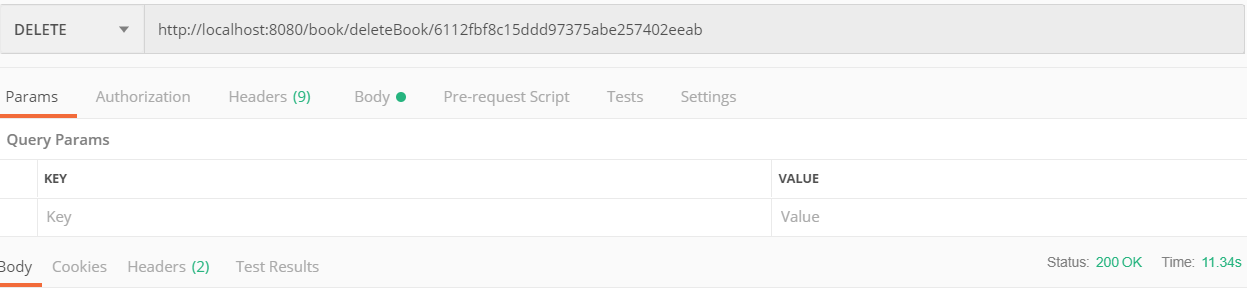
In order to delete a document from CouchDB the following endpoint was set up.

/deleteBook/{id}



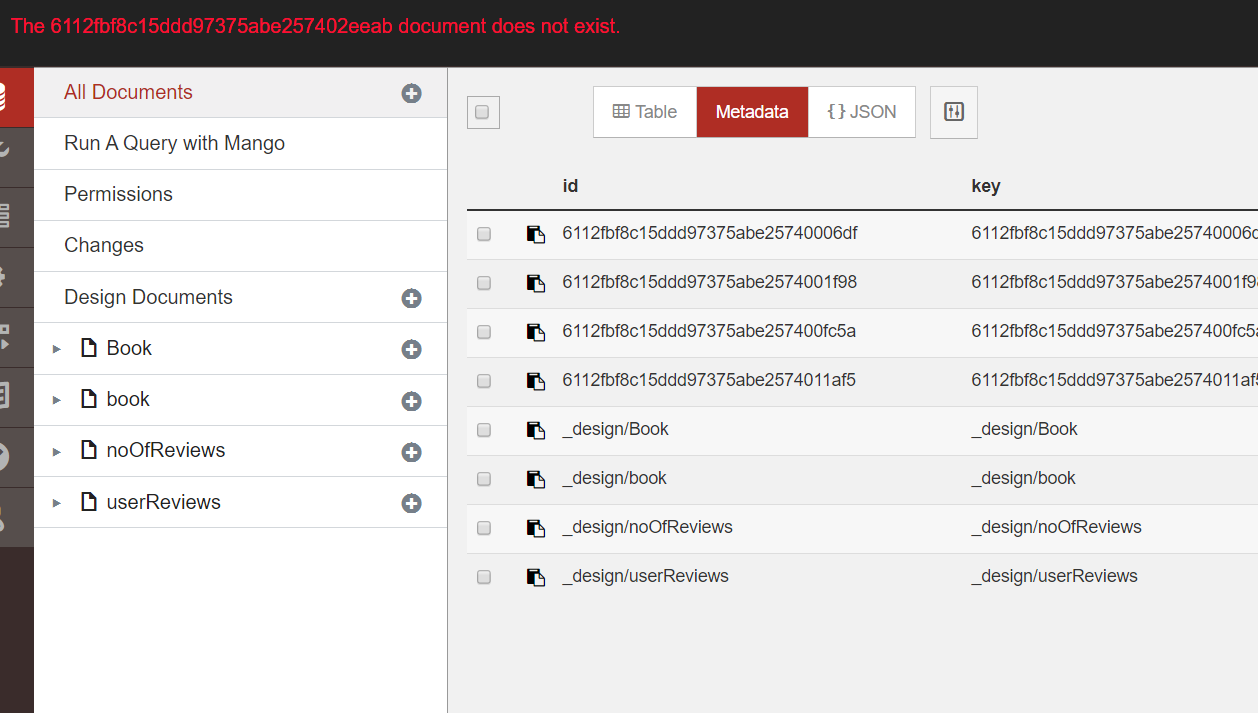
Figure

The delete endpoint takes in just an id parameter in order to delete a document. What is really interesting to note is that Ektorp doesn’t have a delete by id method. Instead it takes an instance of your persistence class. To accomplish this the method had to retrieve the book then pass the book into Ektorps remove method.



Figure

Figure 43 shows the Delete endpoint being tested. The id being used is the same id that was used in section 2.4.1 create. The request received a response code of 200 OK that suggests that the endpoint was hit, and the document was deleted.

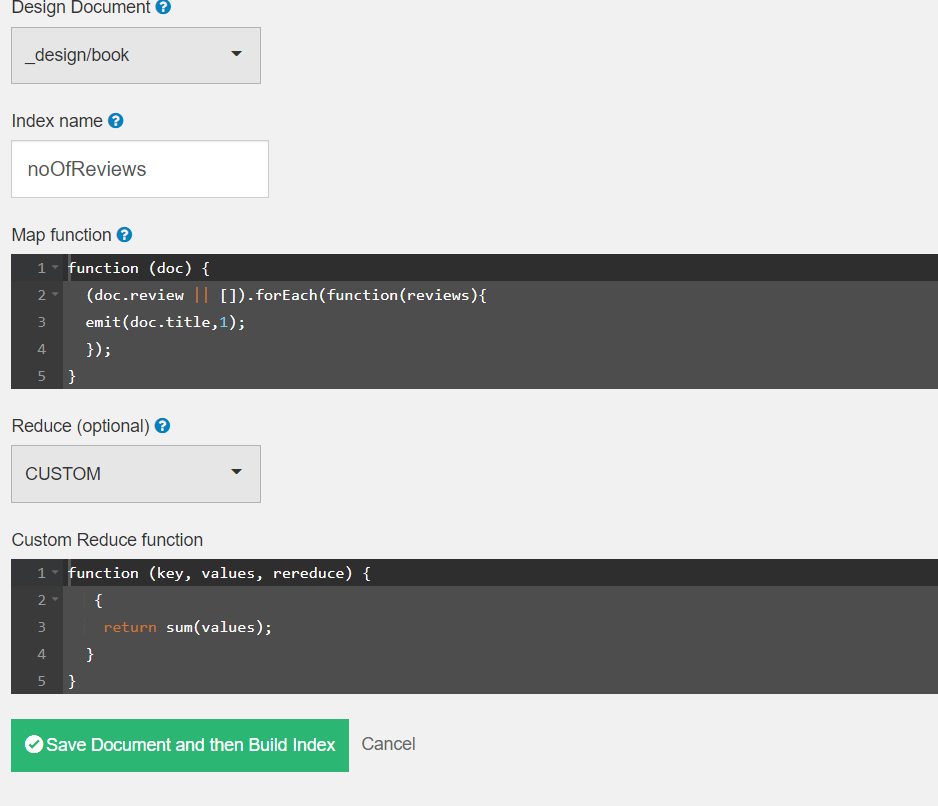


Figure

As can be seen in figure 44 the document has indeed been deleted.

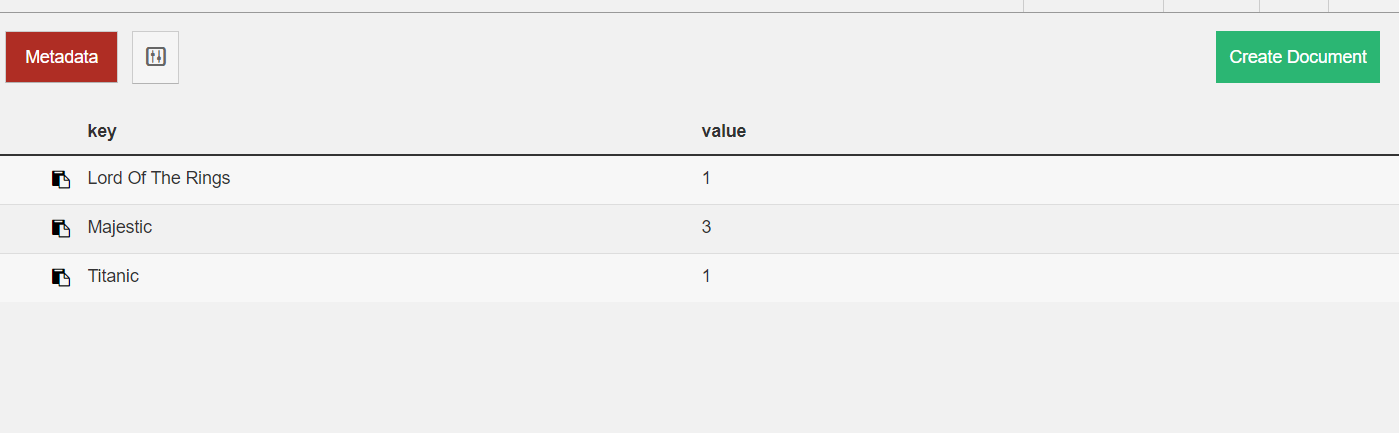
### 2.4.8 Map Reduce

The Review It application also makes use of CouchDB’s map reduce functionality. The map reduce that was created for Review It displays the number of reviews for a specific book. The idea being that when the user is scrolling through the review’s he or she will be aware of how many reviews are present. Here is the map reduce that was created in CouchDB.



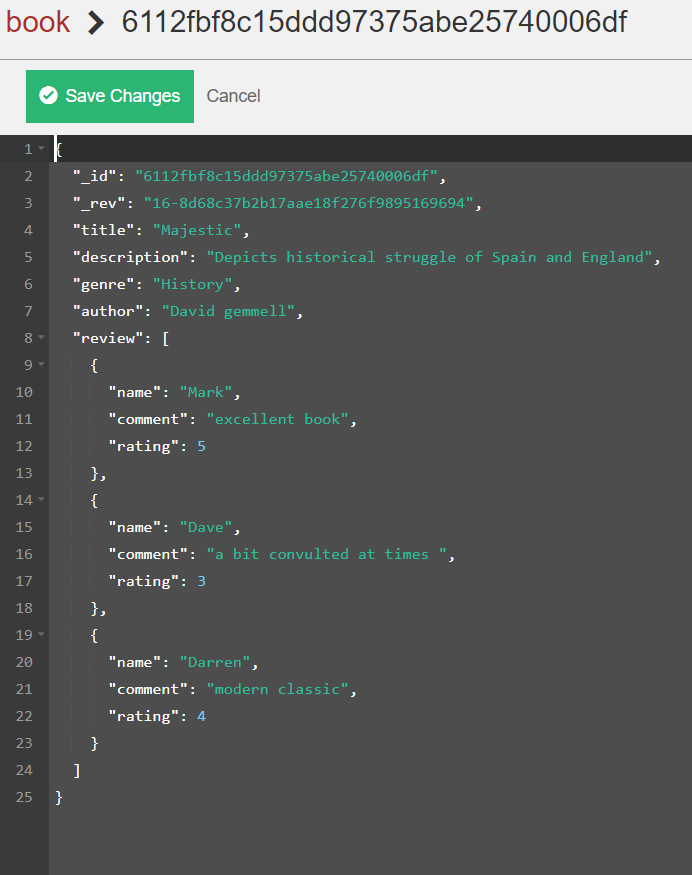
Figure

CouchDB makes it easy to create map reduce functionality by splitting your queries into map and reduce. As is evident from figure 45 theses queries are written in javascript.

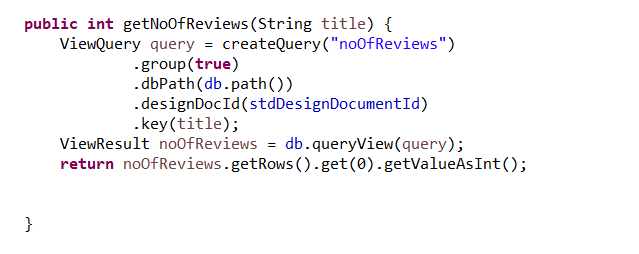


Figure

Figure 46 shows the results of the map reduce query. The left shows the title of the book and the right shows the amount of reviews associated with that book. To further demonstrate that this map reduce has worked. It can be seen in figure 47 that the book Majestic contains a total of 3 reviews.

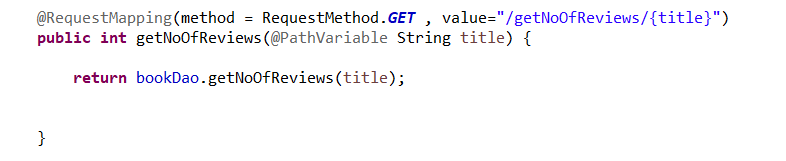


Figure

To access the map reduce functionality from my java web application I had to create a getNoOfReviews method in the DAO layer. The reason it was added to the DAO layer is because it directly interacts with the Datasource. 

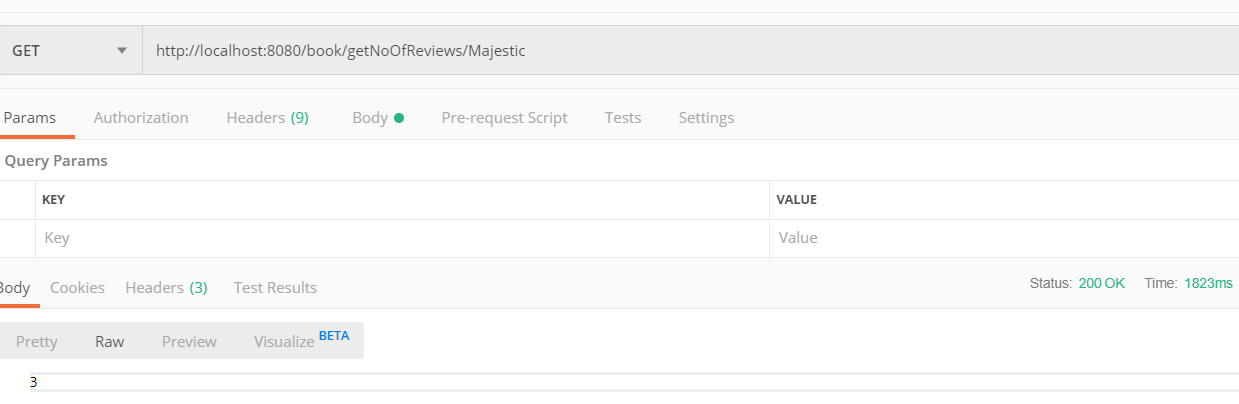
Figure

The method in figure 48 returns an integer value and takes the title of a book as parameter. As can be seen the method reference the map reduce’s noOfReview’s and querys that view. It also assigns the param of title as a key in that query.



Figure

Figure 49 shows the endpoint that was created in order to access the map reduce functionality. As can be seen it takes in a param of title which it gets from the endpoint. It also uses a request type of GET. As can be seen the bookDao dependency now has access to the method that can be seen in figure 48.



Figure

As can be seen from the figure 50 when the route is tested and the Majestic param passed in, the request receives a status code of 200 OK. As can be seen the request returned an integer value of 3. The result of 3 is the expected result this can be seen by comparing the result to figure 46.

# Chapter 3 React Review it’s front end

## 3.1 Introduction To React

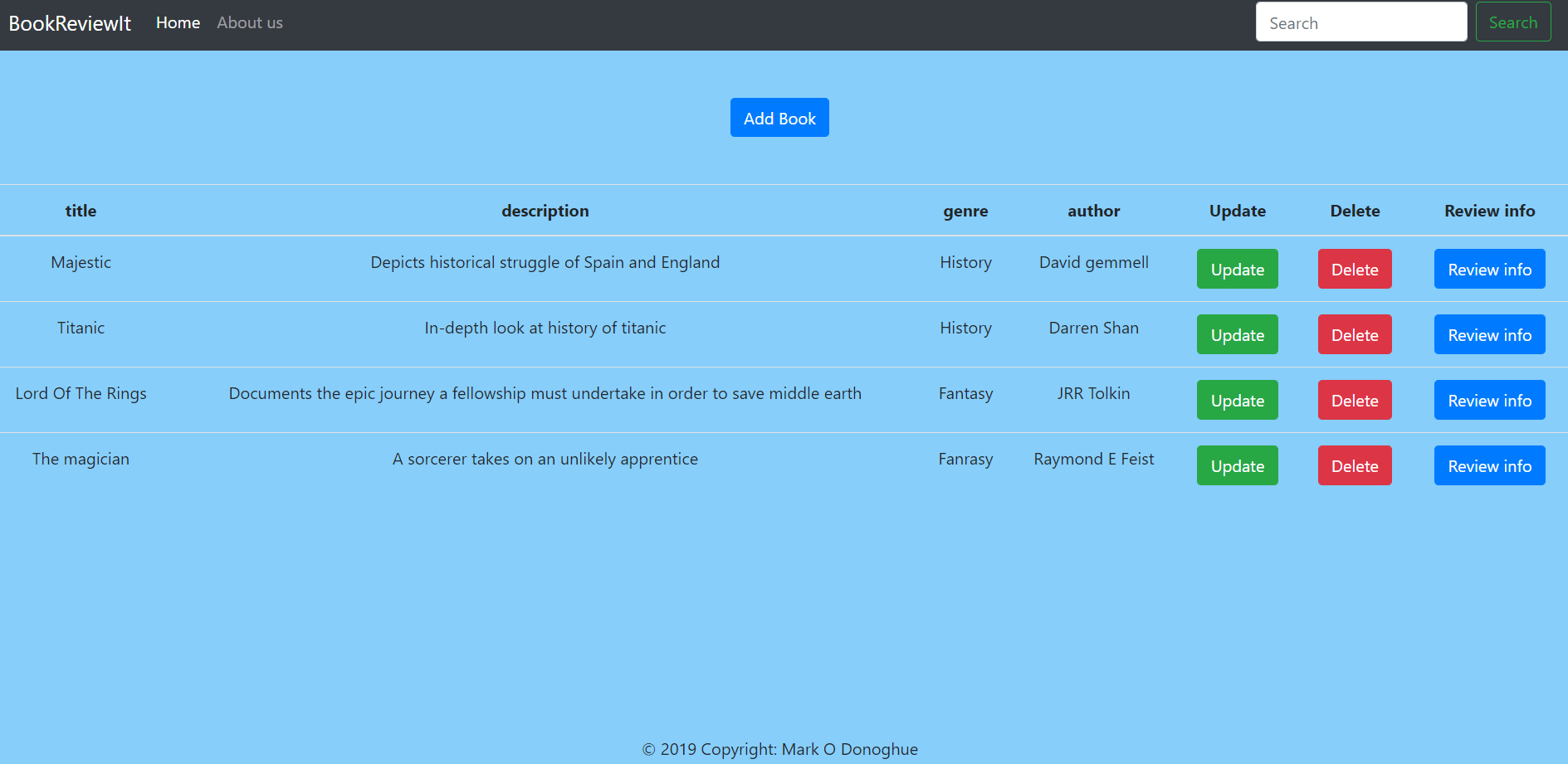
The Review It application was built with a React.js front end. The reason behind choosing this javascript library is because it is one of the most popular javascript libraries available at the moment. To give an example of its popularity React is owned maintained and used by Facebook one of the world’s most popular and known web applications. Another reason for using React.js was that its component based. This means that you split your view into small components. These components when used together create your web page. The advantage of this is that it supports code reusability as these components are 100 percent reusable in other parts of the application. This also promotes code readability, since the components are smaller than a standard page, they are by default more readable. Making it easier for a developer to add functionality or debug existing functionality. Since React runs on port 3000 and the java application works on port 8080 a proxy was created in order to allow the two applications to communicate with each other. The first step after npm installing react was to add a proxy to the project figure 51 shows exactly how this was done. Without use of this proxy the applications wouldn’t have been able to communicate and this would have led to CORS(Cross-Origin Resource Sharing) issues. This problem is written about in the react user documents were they mention “To tell the development server to proxy any unknown requests to your API server in development, add a proxy field to your package.json. Conveniently, this avoids [CORS issues](https://stackoverflow.com/questions/21854516/understanding-ajax-cors-and-security-considerations) and error messages” (React, 2019)



Figure

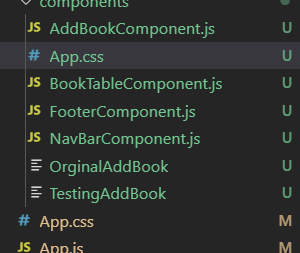
## 3.2 Review It Reacts favourite page

The page that was created for the Review It web application is the favorites page. The goal of this page was to allow a user to add a book. The application will also display information on books that the user has interest in. The favorite page also allows a user to view all the review information associated with a book that they select. The favorite page will also allow a user to add a review, update a review and delete review. It will also use the map reduce functionality this will display the no of reviews for a book that the user selected.



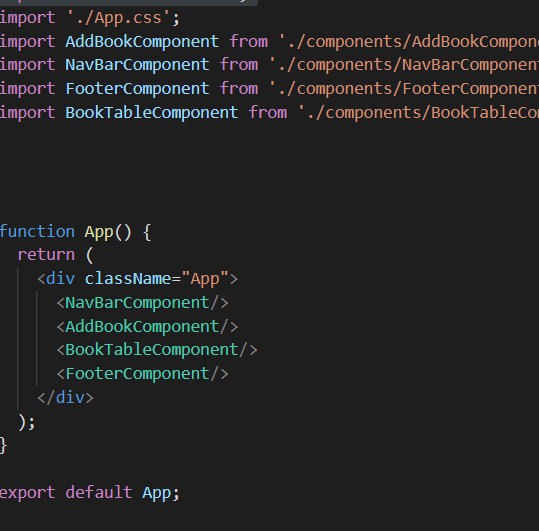
Figure

Figure 52 shows the layout of the favorites page. As can be seen the pages shows details on the books the user is interested in. Each book also contains 3 buttons to update delete and view the review information. A bottom at the top is used for adding new books.



Figure

Figure 53 shows the components that were created in order to display the page from figure 52.

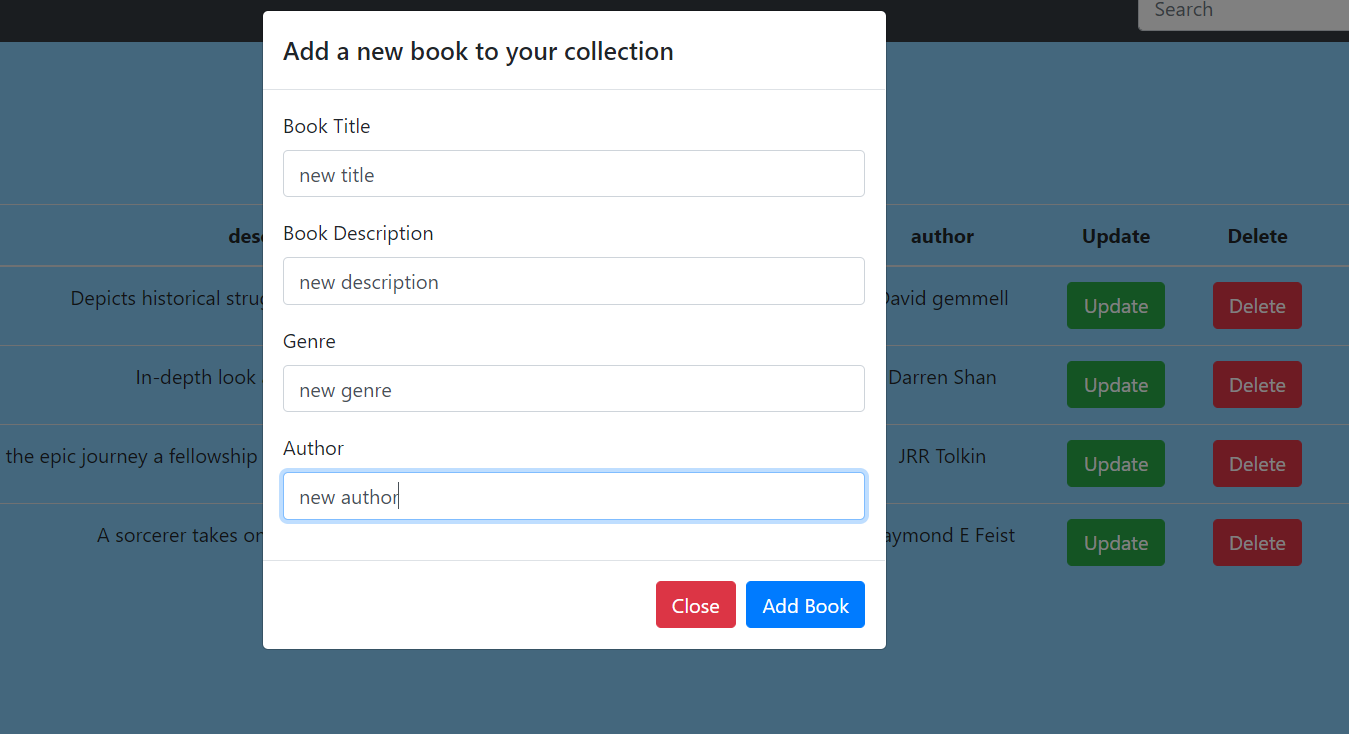


Figure

Figure 54 shows the how these components are used to render the page to the DOM. By doing this you are building your page from components the way that react intended.

## 3.2 Add Book React functionality(Create)

When the Add Book button is clicked the user is presented with the following form.



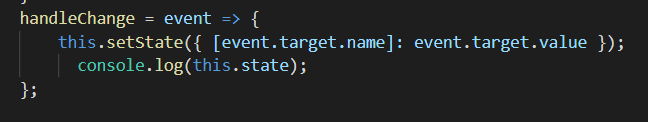
Figure

As can be seen from the form in figure 55 the form has been filled with values. Upon clicking the Add Book button these values update the React state. React state is Reacts way of dealing with information that is of value to the user. State influences how a component reacts and behaves. It makes your components dynamic and interactive in nature. When a state changes the component responds by re-rendering the page. In this way you can create an SPA(Single Page Application). The advantage of this is that the page is interactive in nature and displays current and relevant information to the user. Using the create as an example when the state changes when you add the book. The book you created is immediately and added to the list of favorites without any user interaction.



Figure

Figure 55 shows the state that was created for the addBook component. As these values change through setState the application will be re-rendered. handleChange binds the value from the form to the state. It can be seen in the state in figure 55 and its method is in figure 56.



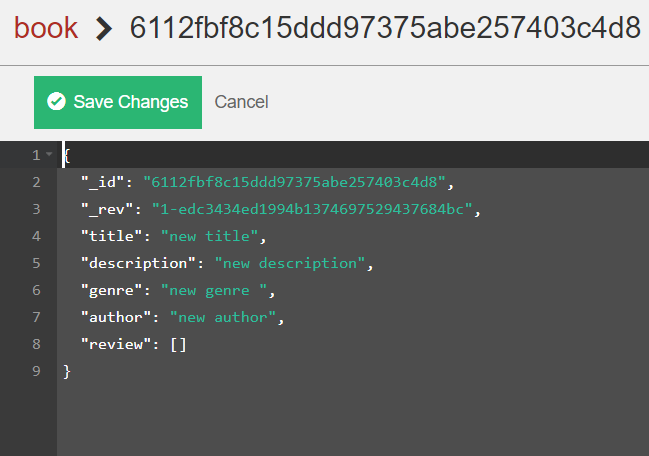
Figure

The add Book button from figure 54 has a pre-defined method called on click. Inside onClick I created a method called addBook. This means that when the Add Book button is clicked this addBook method is called.



Figure

As can be seen in figure 57 the addBook endpoint is being called by the fetch request. No need for the localhost 8080 as that is being specified in the proxy. The method creates variables id title etc and populates them from the react state of the AddBookComponent. Once the request has been sent it sets the state of the used variables to empty. From this point the process from section 2.4.1 begins. But just to show it worked, figure 56 shows the details that were entered into the form.



Figure

Figure 59 show that the details the user provided were saved from the front end through the java application the CouchDB Datasource.

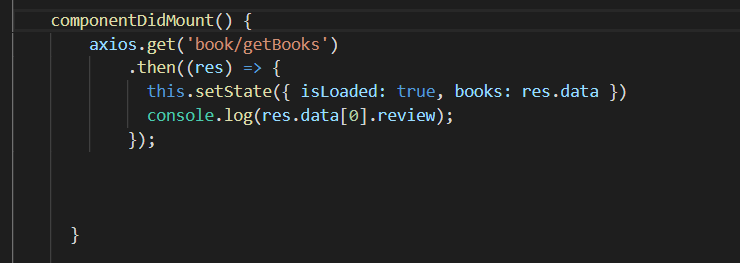
## 3.3 Display Books React functionality(Read)

The read functionality happens immediately when the user visits the favorites page. The user is immediately presented with the all the information regarding his favorite books.



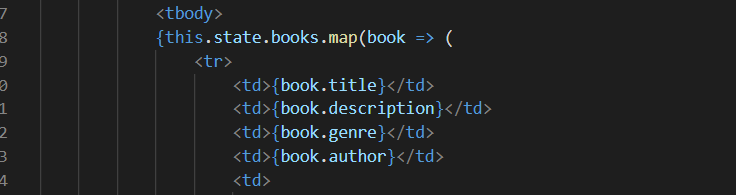
Figure

Figure 60 shows the information that’s coming directly from CouchDB including the book that was created through the UI in section 3.2. This is accomplished without any user interaction through the use of a react lifecycle hook. The lifecycle hook that was used is componentDidMount(). This is a predefined hook of react and what it does is it calls the method after a component is mounted but before the page is displayed to the user “componentDidMount() is invoked immediately after a component is mounted (inserted into the tree).  If you need to load data from a remote endpoint, this is a good place to instantiate the network request.” (React, 2019)



Figure

Figure 61 shows the hook that was used in the Review It application. As can be seen from the image the endpoint that was shown in section 2.4.2 is being used to populate a books object. This is accomplished be setting the state books object to be equal to the response.data.



Figure

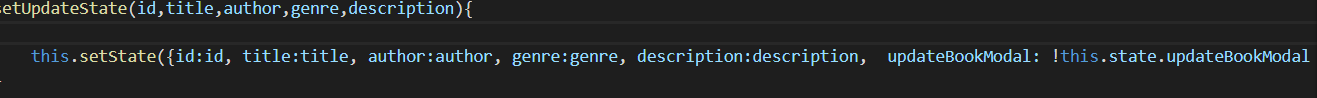
Figure 62 shows how the data is displayed to the user. The books array is mapped onto a book object and the book attributes are displayed in a table. This does this for each individual book till all the documents in CouchDb have been displayed to the user.

## 3.4 Upate Book React functionality(Update)

To update a book, the user selects the update book button. On click of this button a setUpdateState method is called. This updates the state of title, description, genre and author. This will display all information about the book in a Modal (pop up window) and allow the user to update any of the data.



Figure



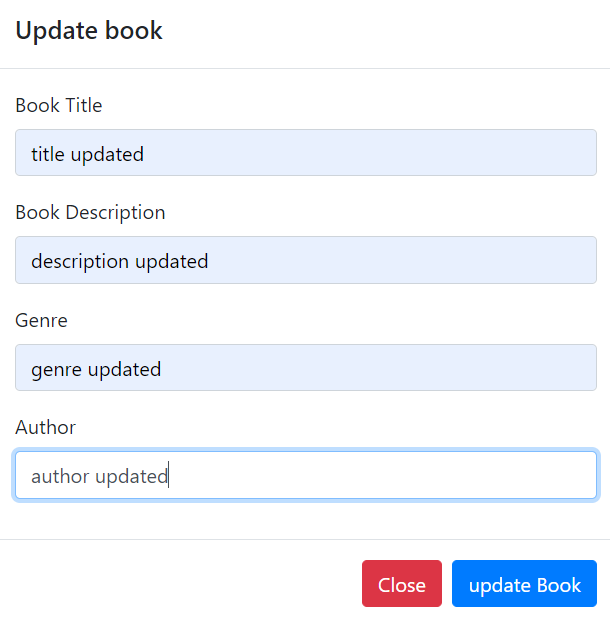
Figure

As can be seen in figure 64 the state is updated and the Modal is updated so that the popup window is displayed to the user.



Figure

Figure 65 shows the form being displayed to the user populated with the book that was created in section 3.2



Figure

Figure 66 shows the new data that has been entered. When the update book button is pressed this calls an update book method.

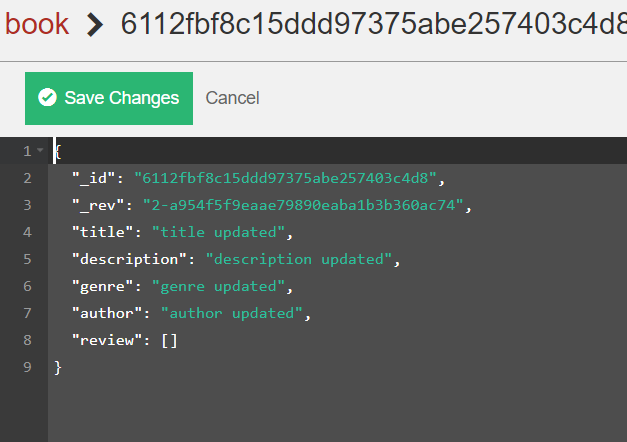


Figure

This method uses the endpoint from section 2.4.4. As is evident in figure 67 the method is a put and the body of the request is populated from the react state. After the request has been made the state is set back to empty.



Figure

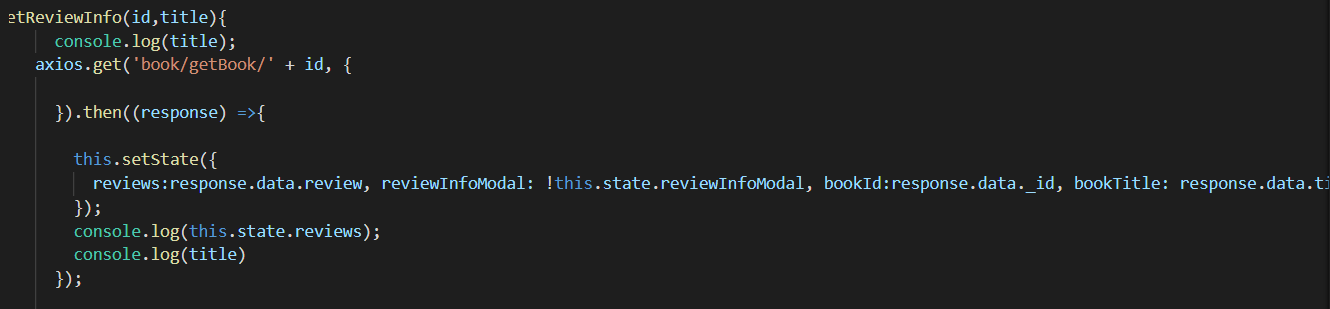


Figure

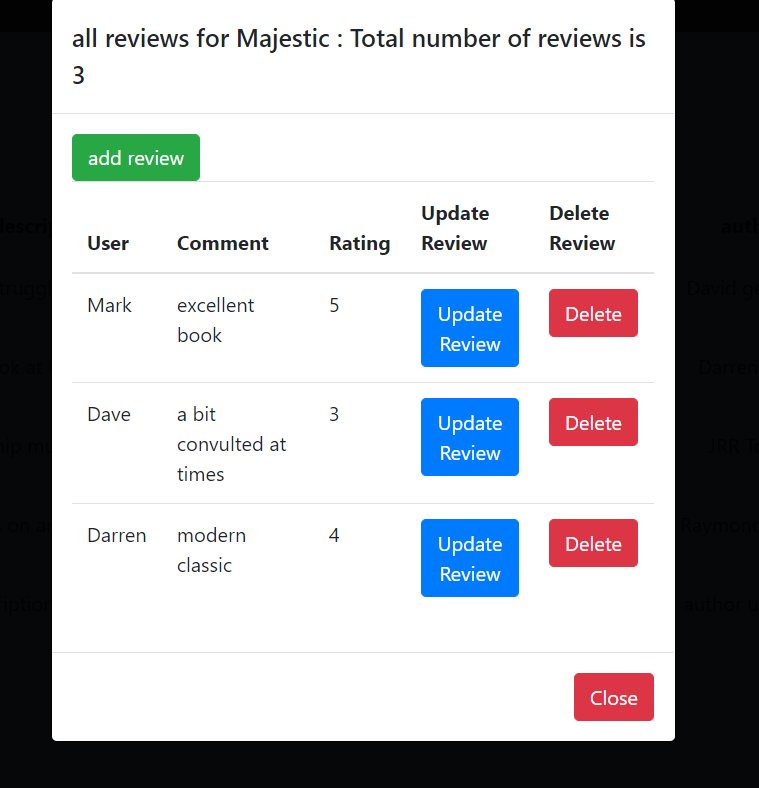
As can be seen in figure 68 and 69 the page has been updated to reflect the update information and CouchDb reflects this update also.

## 3.5 Add Review React functionality (Update, Read 1)

In order to add a review, the user needs to select the review info button see figure 68. After the user makes this selection he is presented with this Modal. This Modal is populated through the read 1 endpoint from section 2.4.3.

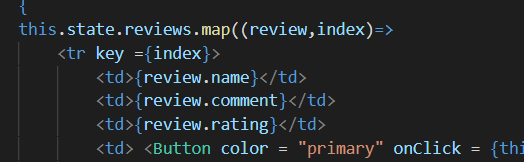


Figure

Figure 70 shows the endpoint being used and the response from that endpoint is being used to alter reacts state. Most notably a reviews array is being populated with the response review information. This reviews array is used to populate a review information Modal. 

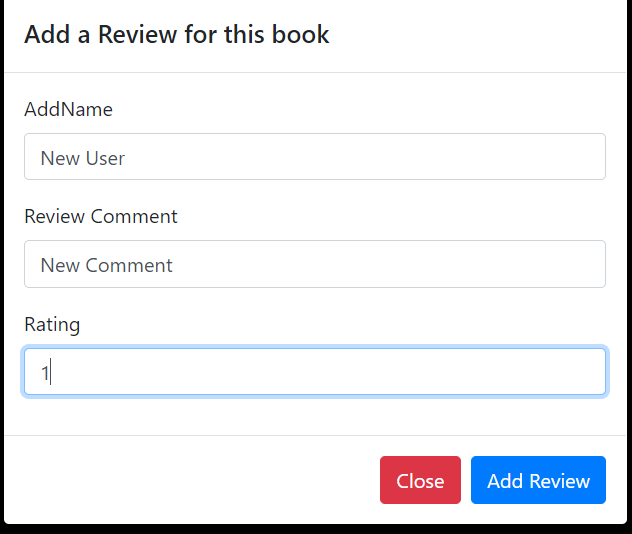
Figure

Figure 71 shows how the review information is displayed to the user.



Figure

Figure 72 maps the reviews objects onto a review object and displays each review in a table to the user. It also tracks the index of each review which is key in allowing update and delete functionality on reviews. Now that the read 1 has been shown let’s add a review to the book that was created in section 3.2.



Figure

Figure 73 shows the form the user sees when adding a new review. The form has been already filled out with the data we want to test the functionality with.



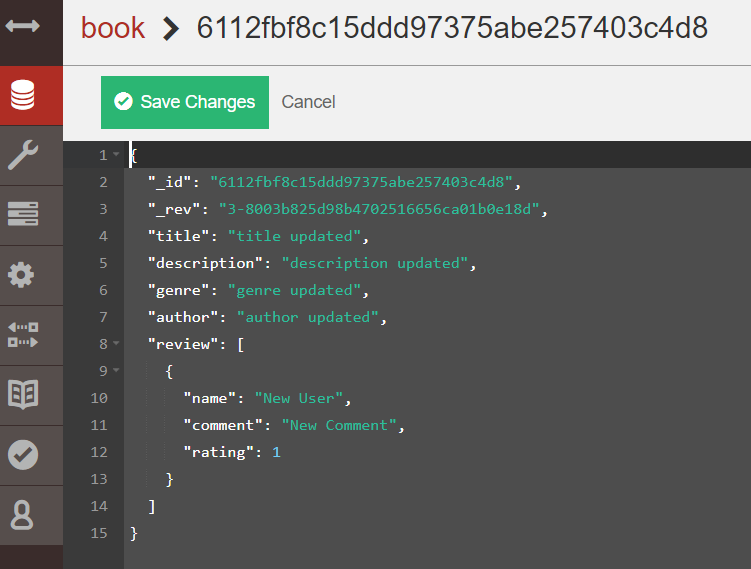
Figure

Figure 74 shows the endpoint that was shown in section 2.4.5. It is a PUT request and its body is populated through react state. After the request has been processed the state is reset.



Figure 75

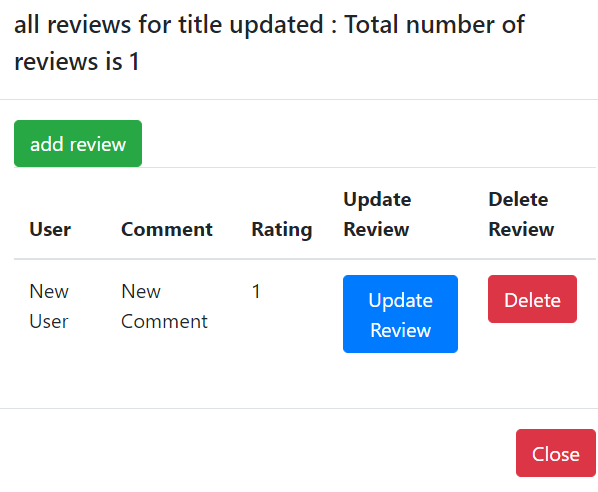
Figure 75 shows that the review has been added in React.



Figure

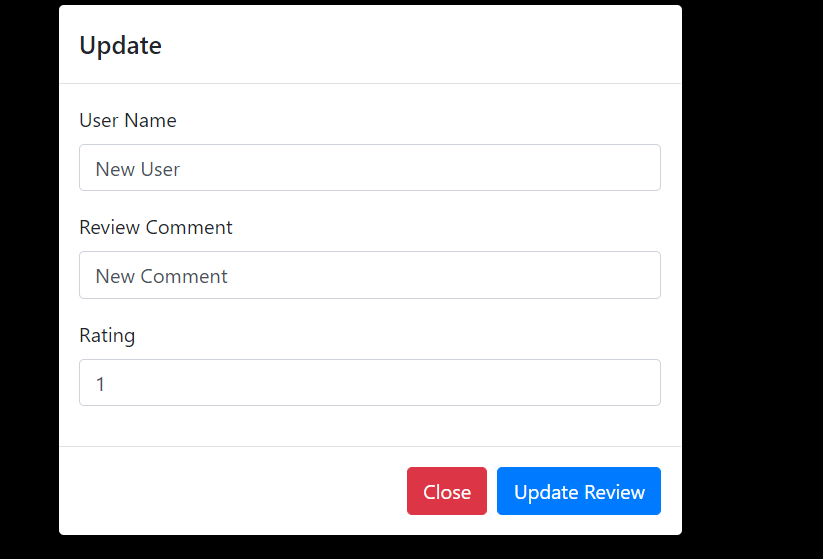
Figure 76 proves that the document has been updated and that the review has been added.

## 3.6 Update Review React functionality(Update)

In order to update a review, the update review button from figure 77 is selected. 

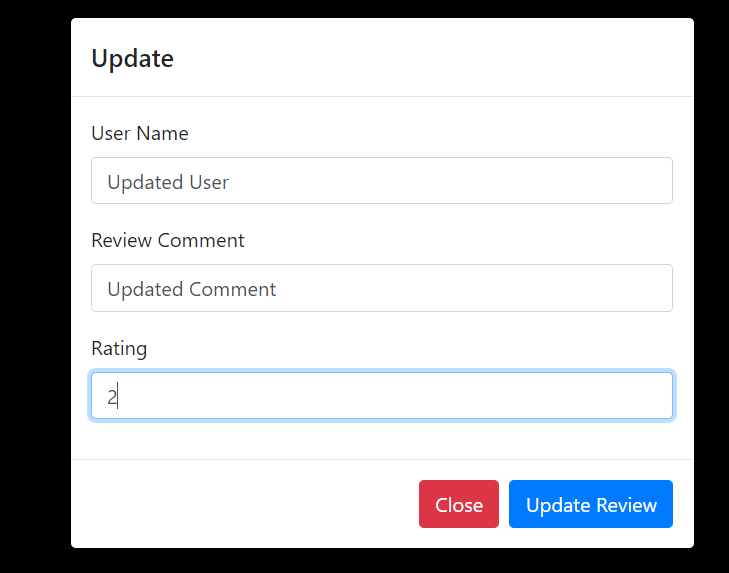
Figure

After the button has been clicked the user is presented with the current review information.



Figure

Now to update the form and click the update review button.



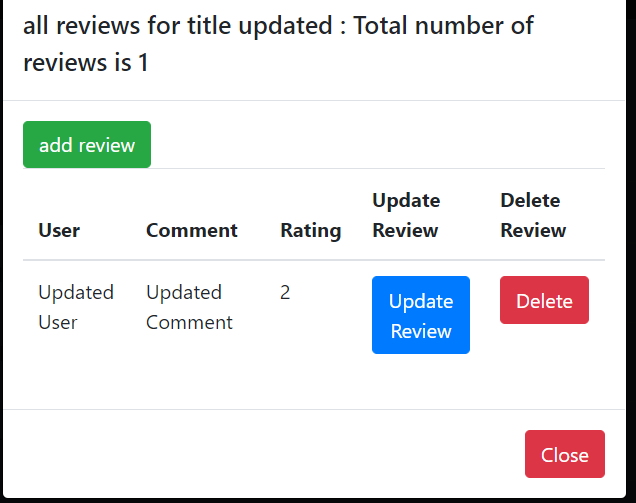
Figure

Figure 79 shows the new details that we wish to update the review with. On click of the update review the updateReview method is called.



Figure

Figure 80 shows that the endpoint from section 2.4.6 is being used. We can also see that the request is a PUT method and that the body is being populated from the React state. The index, meaning position in the array is also passed into the endpoint and this is also populated from the state.



Figure

Figure 81 shows that the frontend has been updated with the correct information.

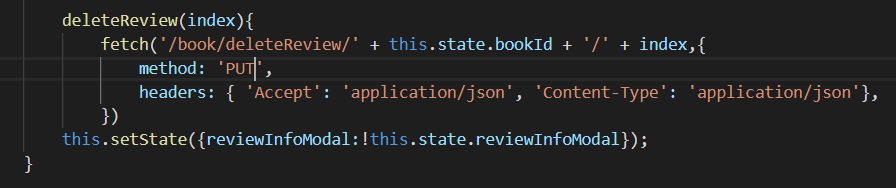


Figure

Figure 82 shows that the CouchDB DataSource has been updated with the correct information.

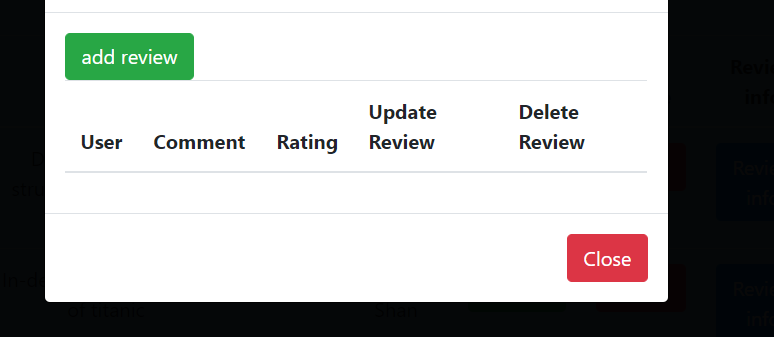
## 3.7 Delete Review React functionality(Update)

To delete a review from a book the delete button from figure 81 is selected. When the button is pressed the deleteReview method is called.



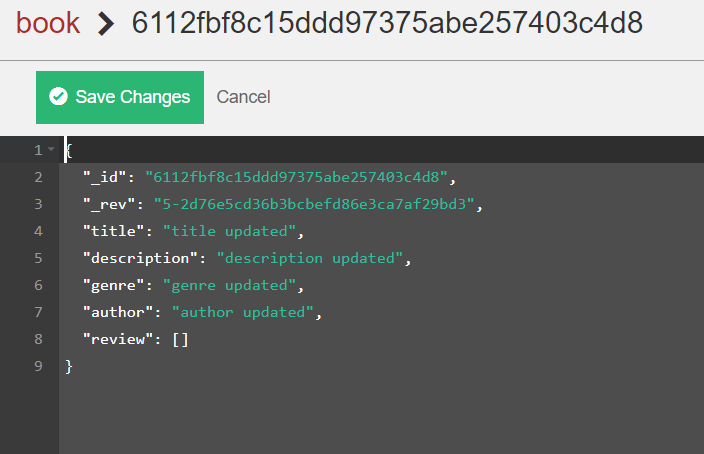
Figure

Figure 83 shows that the endpoint from section 2.4.7 is being used. The bookId is being populated from state and the index is being supplied as a parameter to the method.



Figure

Figure 84 shows that the review information is no longer present in the UI.

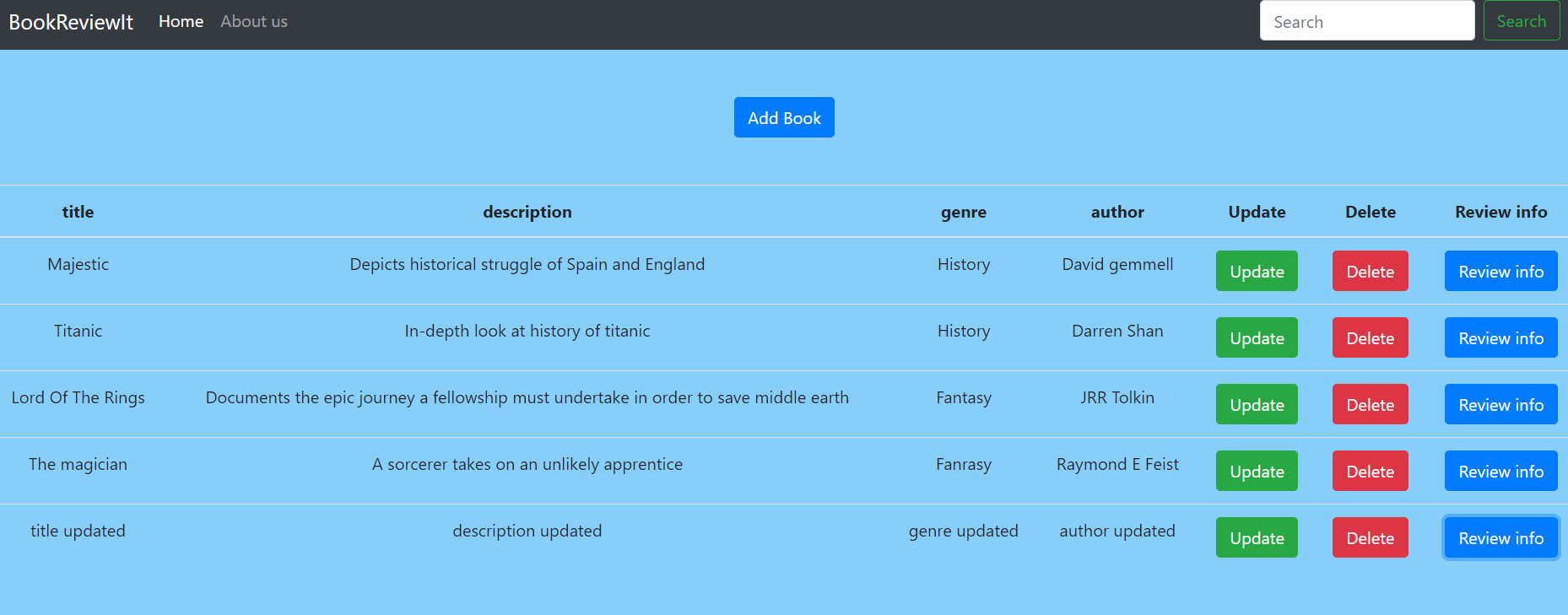


Figure

Figure 85 proves that the review information has been updated and is no longer present in the document.

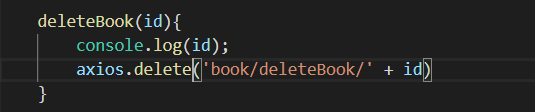
## 3.8 DELETE Book React functionality(Delete)

A book can be deleted from the favorites page. To perform the delete the user must click on the delete button.



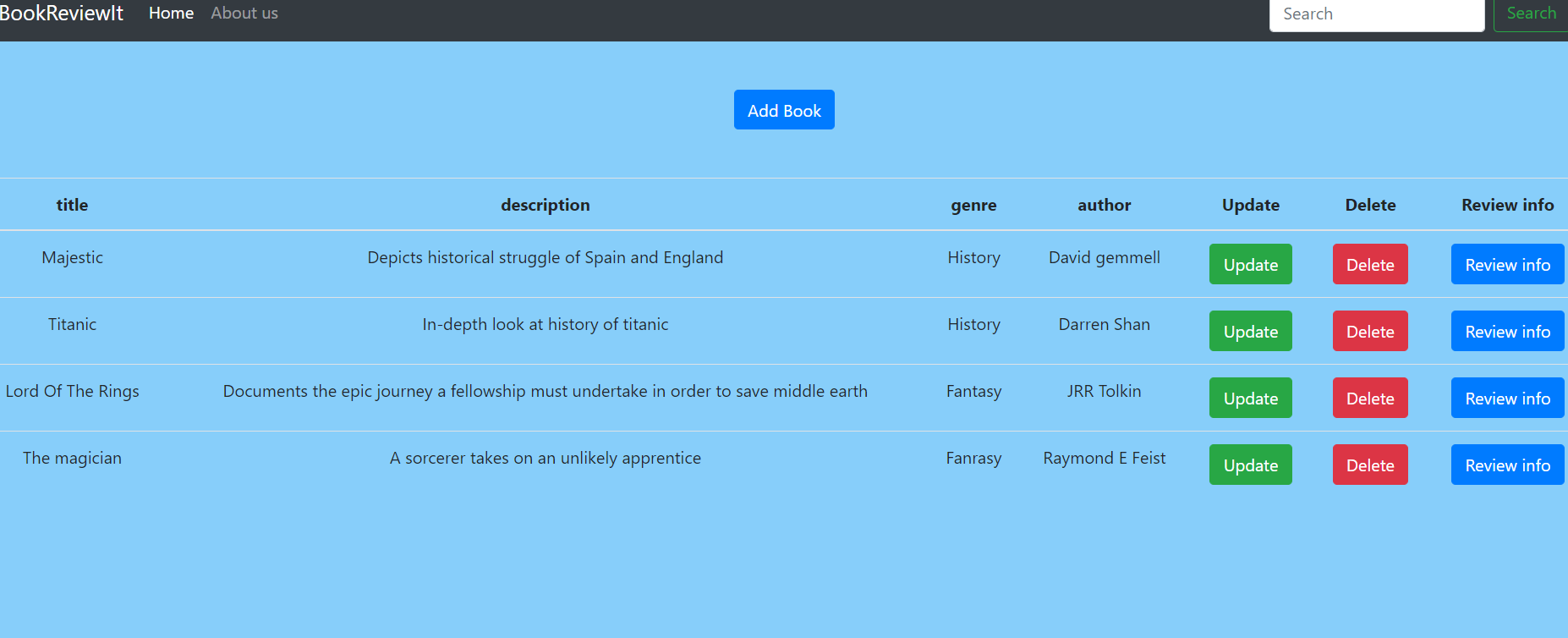
Figure

Figure 86 shows were the delete button is located. Once the button has been pressed the deleteBook method is called.



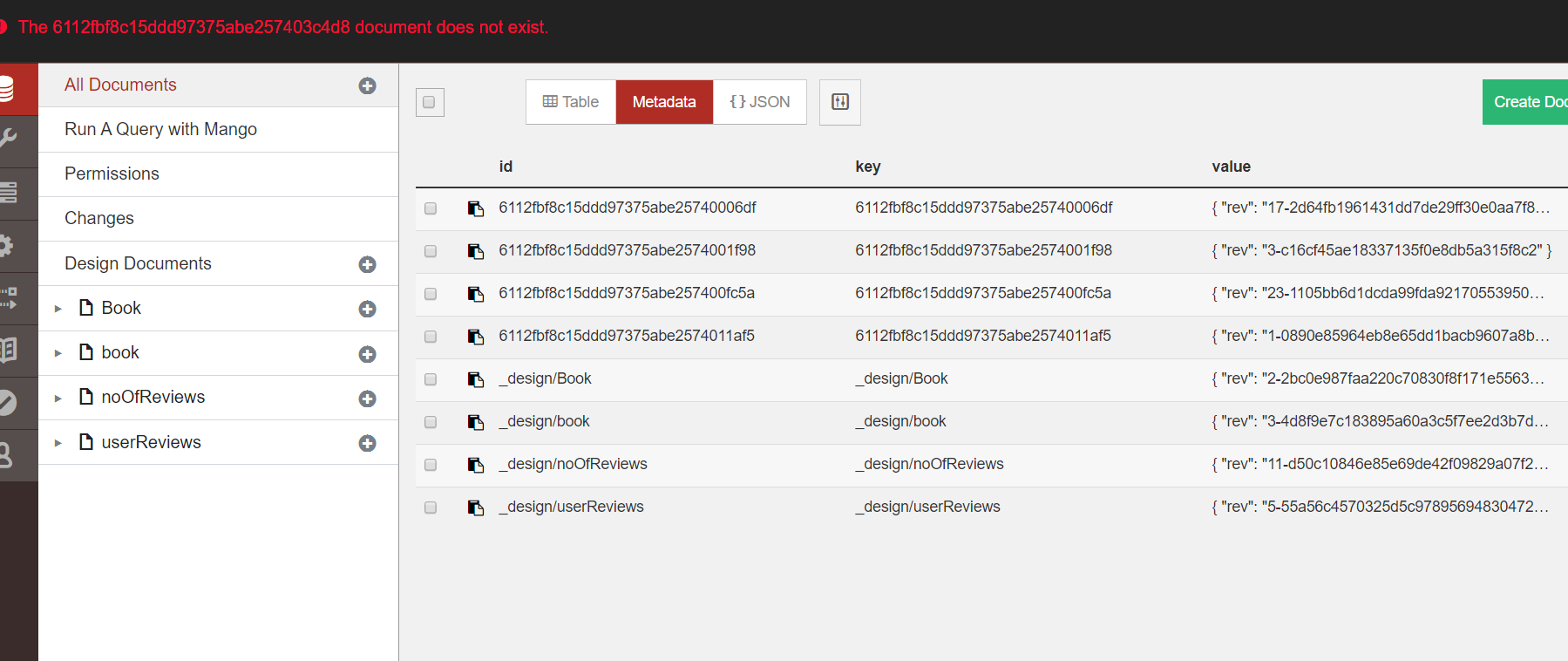
Figure

Figure 87 shows the method. It takes in the books id as a parameter then appends it to the endpoint from section 2.4.7.



Figure

Figure 88 shows that the book has been deleted from the front end.

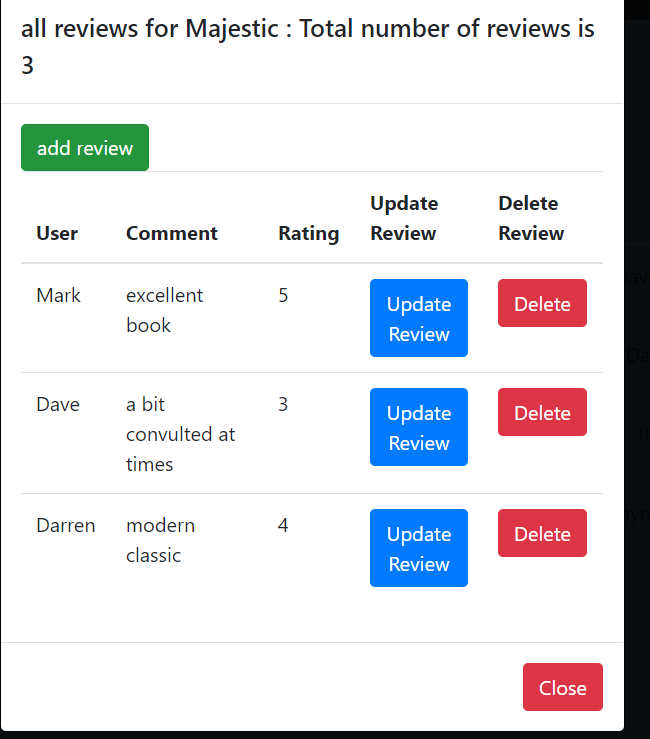


Figure

Figure 89 proves that the document has been deleted.

## 3.9 Number Of reviews React functionality(Map Reduce)

Finally, the map reduce functionality can be seen when a user selects the review info button from figure 88. Upon clicking the button the user is presented with the review information. In the header of that Modal a number of reviews is being displayed to the user.



Figure

Figure 90 shows that the number of reviews for the majestic book is 3 which is correct. This is accomplished through the getreviewInfo method.



Figure

Figure 91 not only shows how the review information is being populated but shows an axios get request to the map reduce endpoint from section 2.4.8. This then populates the noOfReviews attributes with the response from request. This attribute is then used to display that number to the user as seen in figure 90.

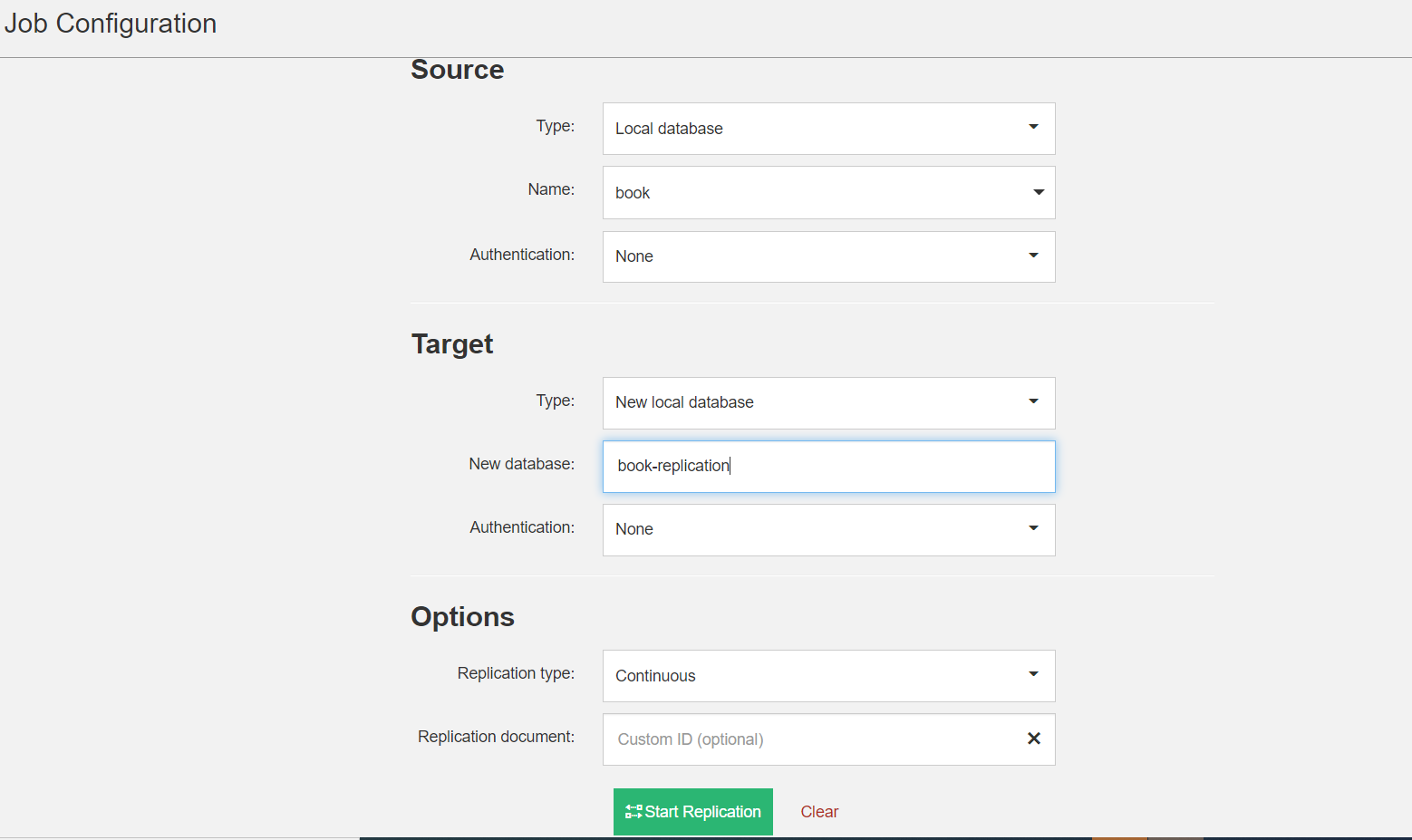
# Chapter 4 Data Replication and Conclusion

## 4.1 Data Replication

As was mentioned earlier replication is easy to achieve with CouchDB. There are 2 forms of replication continuous replication and once off replication.

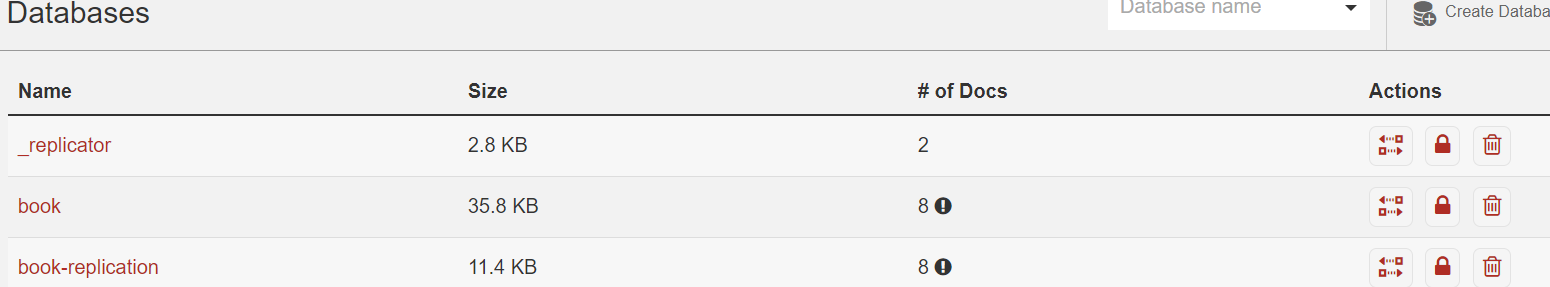
### 4.1.1 Continuous Replication

Continuous replication means that when any change is made to the database CouchDB will attempt to replicate this change onto the replicated database. In order to perform continuous replication a user just has to fill in the form in figure 92.



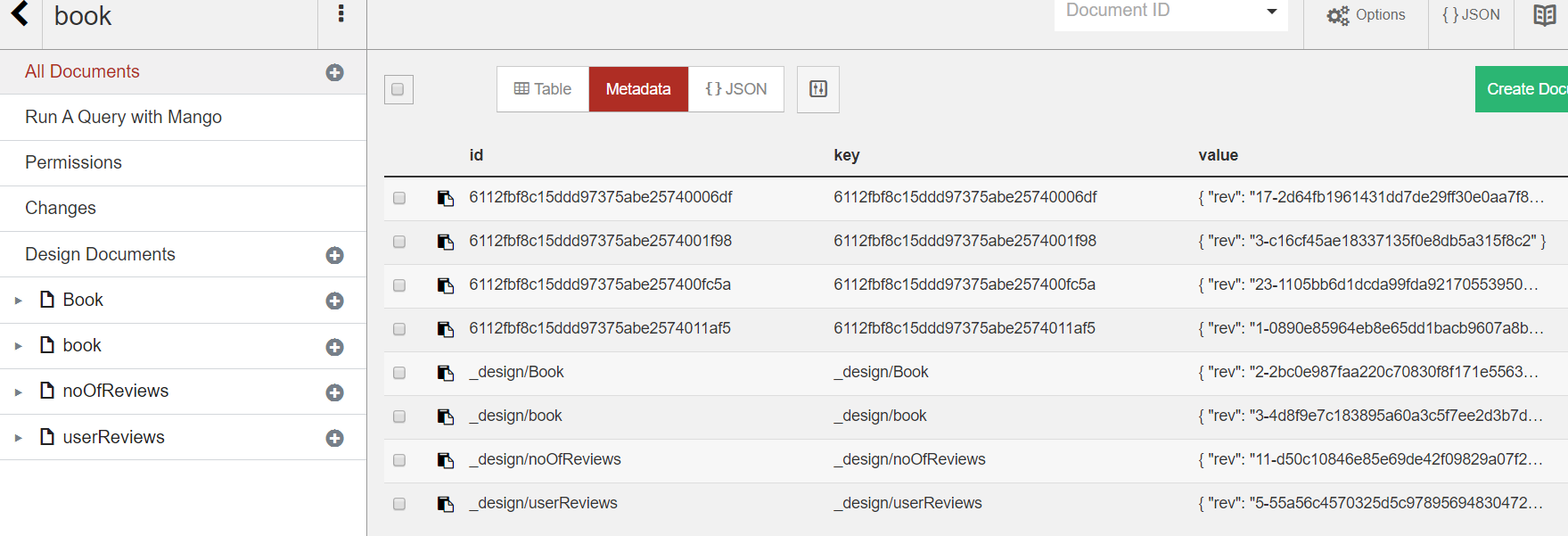
Figure

Figure 92 shows the that the replicated database is given a name of book-replication.

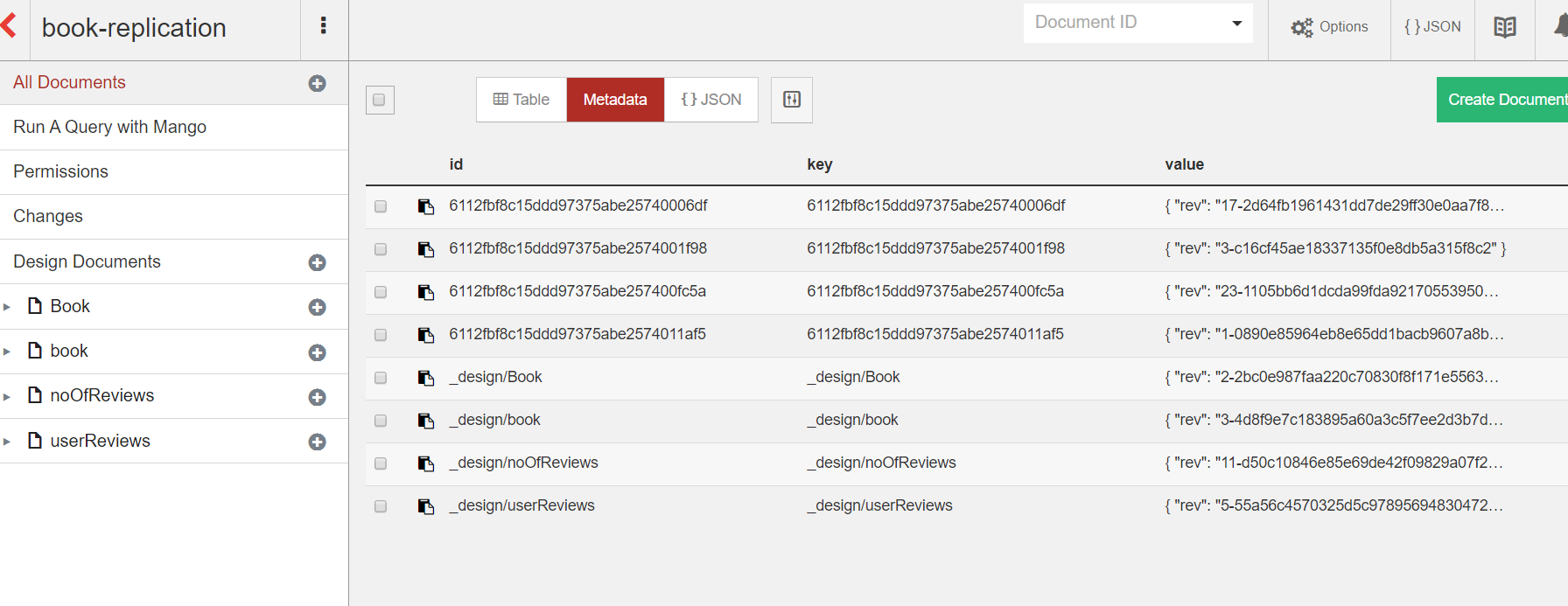


Figure

Figure 93 shows the replicated database has been created and figure 94 shows the contents of the original database and figure 95 shows the contents of the new replicated database.

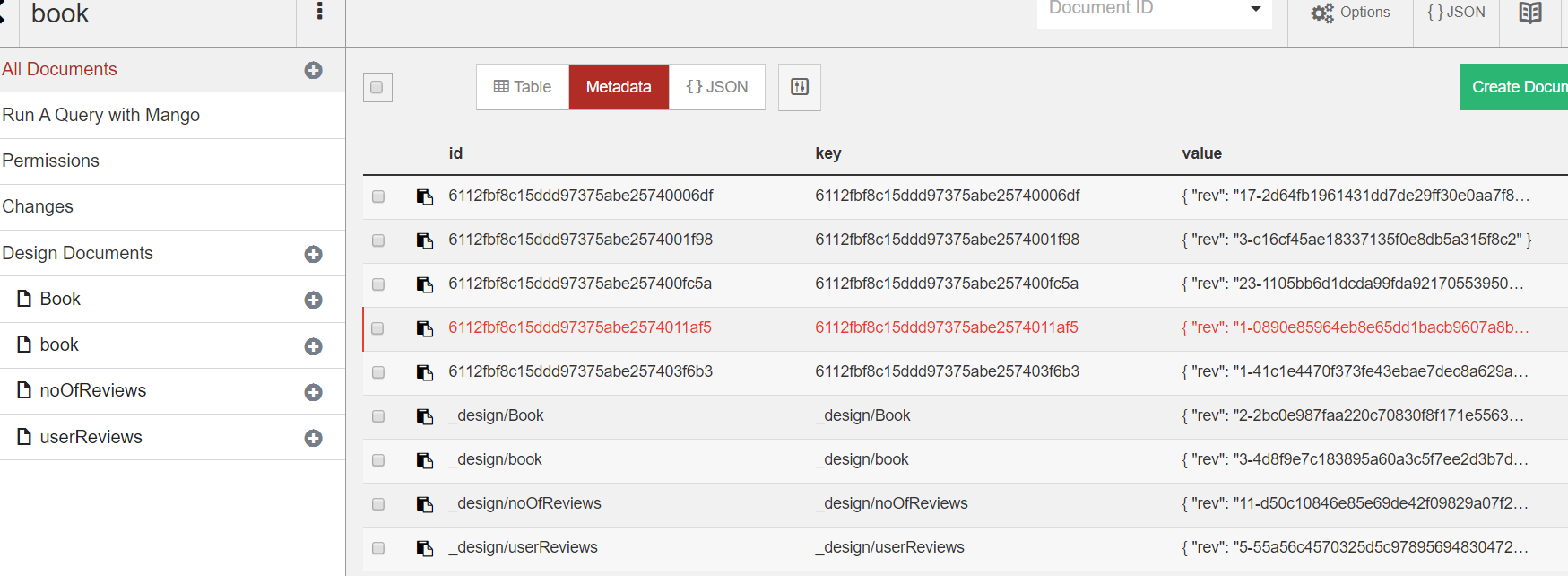


Figure

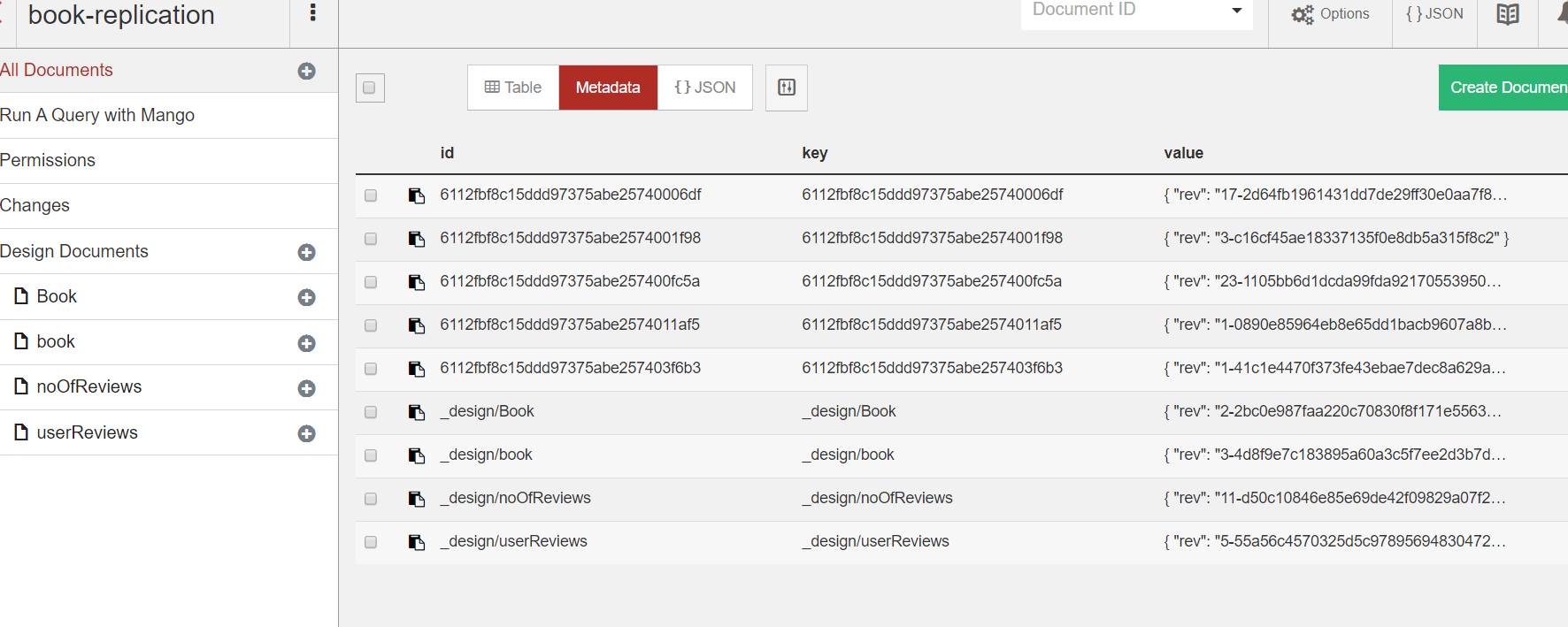


Figure

As can be seen from both images the contents match exactly. However, the only way to prove that this is in fact continuous replication is to add a new document through the application and see those the replicated database picks up the change.



Figure

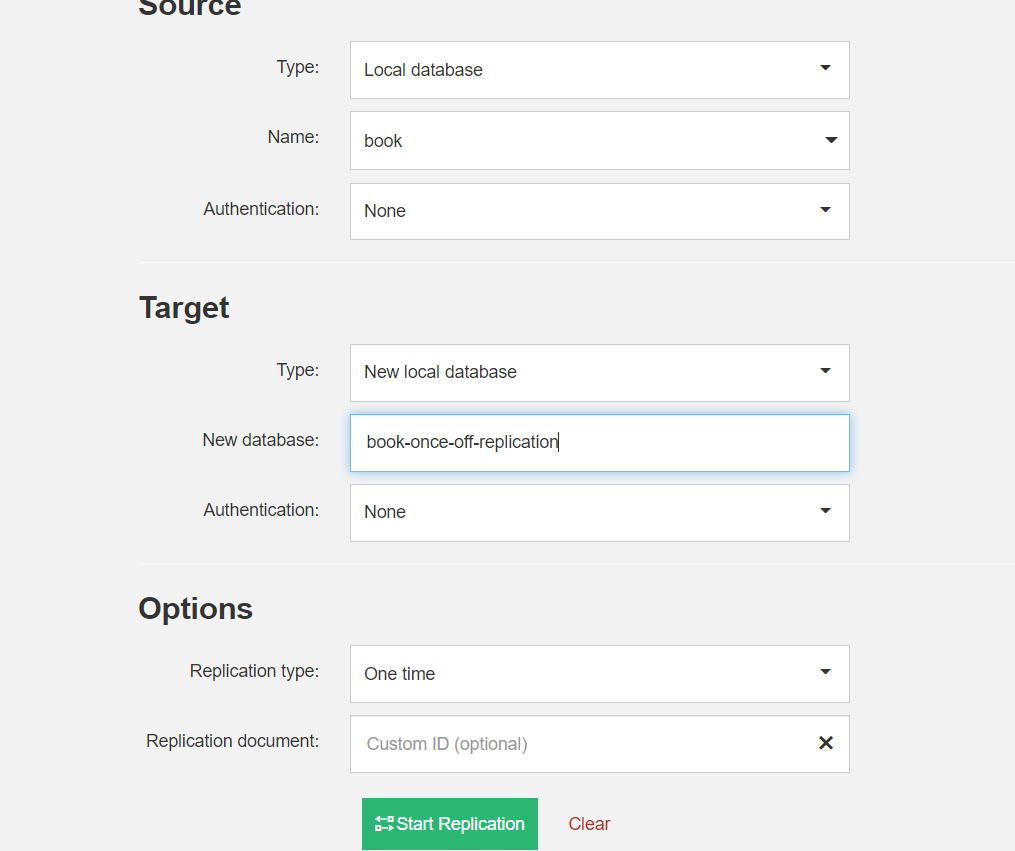


Figure

Figure 96 represents the original database and 97 the replicated one as can be seen continuous replication has been achieved as both databases have a new document, but it was only added to the book one.

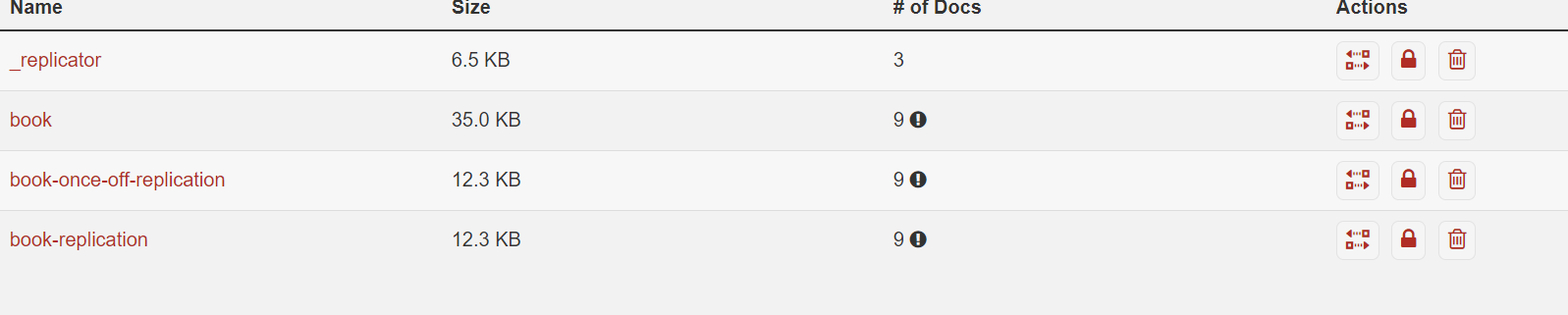
### 4.1.2 Once Off Replication

This form of replication means that only the current state of the database will be replicated, and no changes afterwards will be replicated. The following form was filled out to achieve this replication.



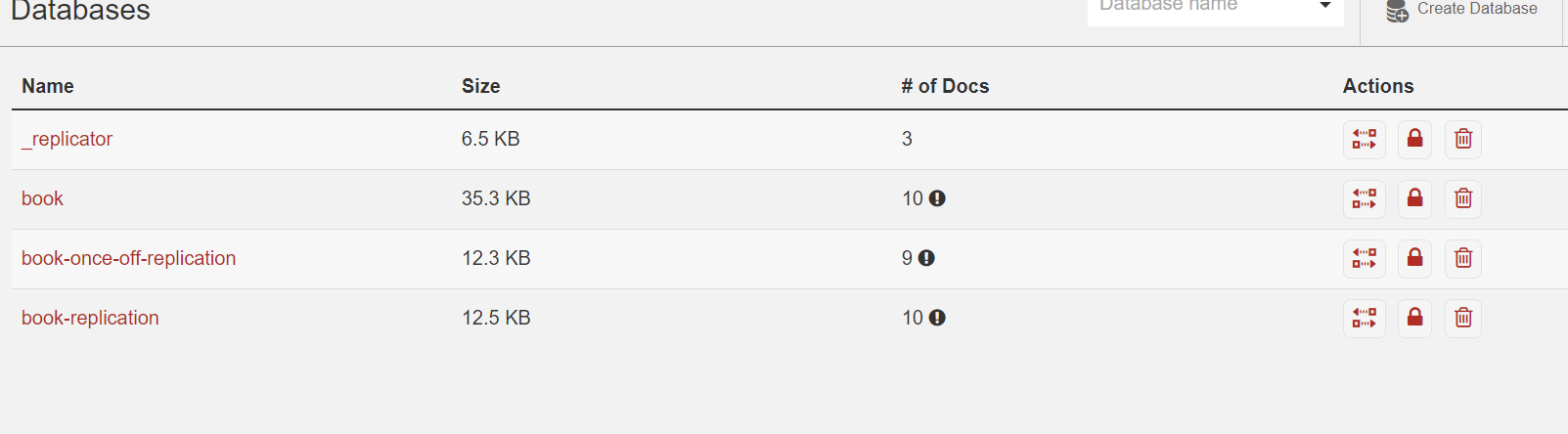
Figure

After this form has been filled out the database is replicated.



Figure

Figure 99 shows that the replication has occurred, and the number of documents shows that it matches the current state of the book database. Now if the book database is updated through the application the once off replication shouldn’t get anymore documents but the continuous one should.

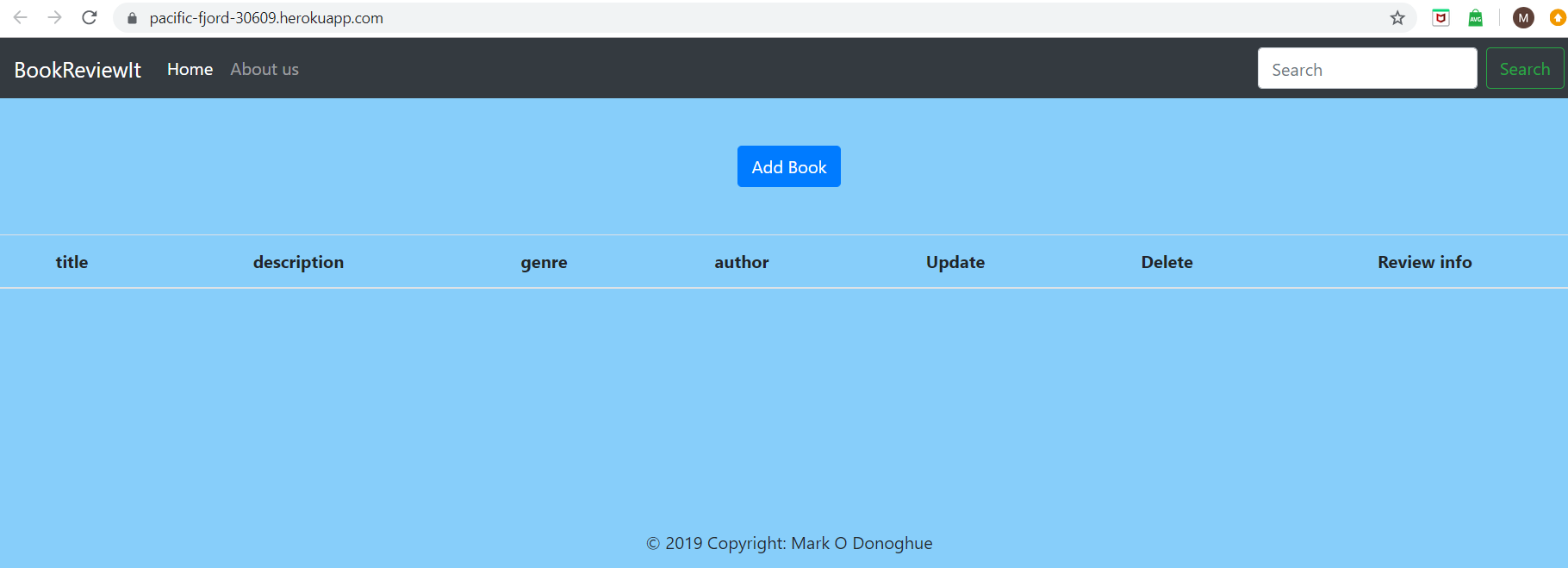


Figure

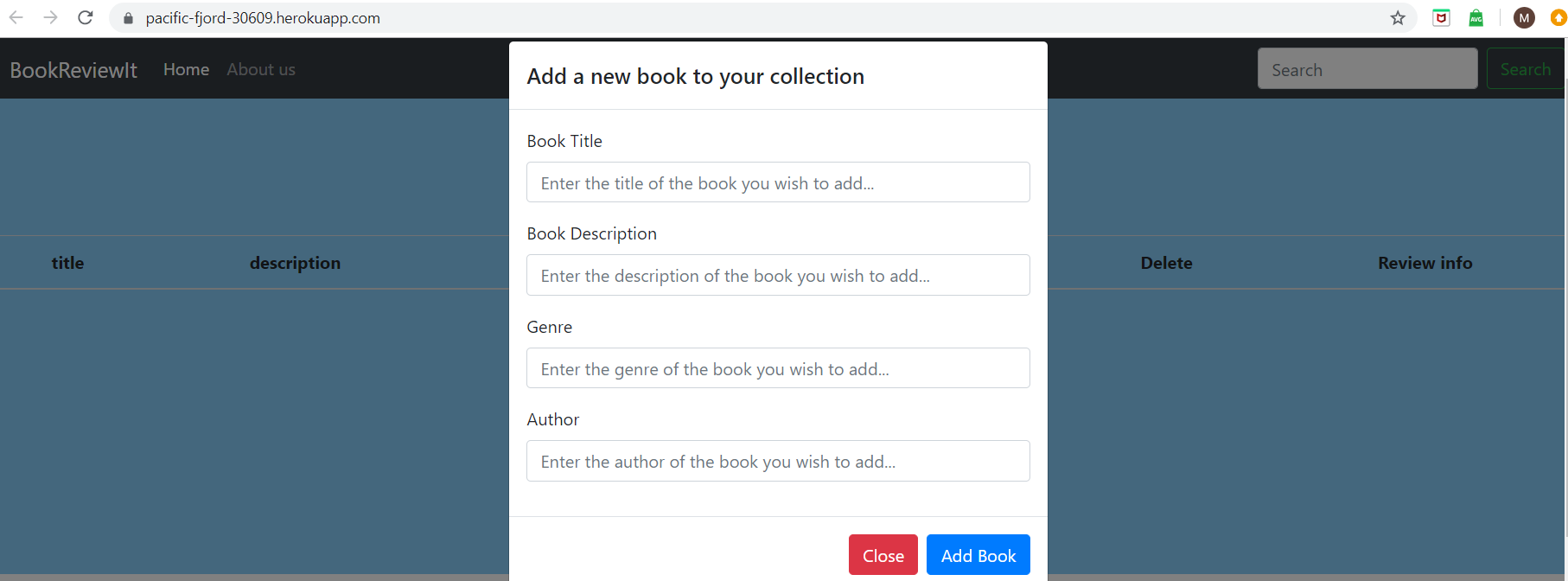
Figure 100 proves that this is the case and that continuous and once off replication have been achieved.

## 4.2 Deploying The React Application To heroku

The React application was deployed to Heroku to simulate the next steps for an application. After deploying the application with Heroku I was given the following url <https://pacific-fjord-30609.herokuapp.com/>. Unfortunately, since the java web application is running locally the application can’t be populated with data from my database or perform any of the CRUD functionality locally. Figure 101 shows the the application hosted on the web. It looks a bit barebones but that is because it is a data driven page without data but the react functionality is still working.



Figure



Figure

As evidenced from figure 101 and 102 the functionality remains but the data is missing.

## 4.3 Conclusion

Overall Id rate CouchDB as an good choice of datasource. Its schema free, real life representation of data made it very easy to manipulate and work with. Its use of a rest interface means that the basic CRUD functionality can be carried out without the use of an API. However, this is BASIC functionality and the way I interacted with the review objects could not have been accomplished with this functionality alone. CouchDb also allows for a wide range of drivers however in comparison to a datasource like Oracle I found the documentation and community support lacking. For java Ektorp was the most documented of the 4 available drivers however as mentioned this documentation doesn’t match up to standards set by other databases. Having said that I found Ektorp easy to configure and work with and provided easy implementation of CRUD functionality. However, I found the way it performs delete operations to be perplexing. The fact that you had to pass in an entire instance of the document as opposed to just the id really isn’t a user-friendly way of providing this functionality. Overall, I was happy with the full stack application that was created and CouchDB would definitely be a database that I would consider in the future.

# References

Avraham, S. B., 2017. *what is rest.* [Online]   
Available at: https://medium.com/extend/what-is-rest-a-simple-explanation-for-beginners-part-1-introduction-b4a072f8740f  
[Accessed 30 October 2019].

Brown, M., 2012. *Getting Started with CouchDB.* 2nd ed. Sebastopol: O'Reilly Media, Inc..

Danylko, J., 2016. *DZone.* [Online]   
Available at: https://dzone.com/articles/what-is-loose-coupling  
[Accessed 2 November 2019].

db-engines, 2019. *couchDB.* [Online]   
Available at: https://db-engines.com/en/system/CouchDB  
[Accessed 24 October 2019].

Gilbert, S. & Lynch, N., 2012. Perspectives on the CAP Theorem. *Computer,* pp. 30-36.

Krill, P., 2009. *CouchDB emerging as a top choice for offline Web apps.* [Online]   
Available at: https://www.infoworld.com/article/2629886/couchdb-emerging-as-a-top-choice-for-offline-web-apps.html  
[Accessed 31 October 2019].

Lundgren, H., 2017. *Ektorp Reference Documentation.* [Online]   
Available at: https://helun.github.io/Ektorp/reference\_documentation.html  
[Accessed 1 November 2019].

Putano, B., 2019. *A Look At 5 of the Most Popular Programming Languages of 2019.* [Online]   
Available at: https://stackify.com/popular-programming-languages-2018/  
[Accessed 2 Nov 2019].

React, 2019. *create-react-app.* [Online]   
Available at: https://create-react-app.dev/docs/proxying-api-requests-in-development/  
[Accessed 2 November 2019].

React, 2019. *React.Component.* [Online]   
Available at: https://reactjs.org/docs/react-component.html#componentdidmount  
[Accessed 2 November 2019].

Slater, N., Jan, L. & Anderson, J. C., 2010. *CouchDB: The Definitive Guide.* 2nd ed. Sebastopol: O'Reilly Media, Inc..

Spring, 2019. *Spring.* [Online]   
Available at: https://docs.spring.io/spring/docs/4.2.x/spring-framework-reference/html/orm.html  
[Accessed 1 November 2019].

Tiobe, 2019. *TIOBE INDEX.* [Online]   
Available at: https://www.tiobe.com/tiobe-index/  
[Accessed 1 November 2019].