Introduction to
NoSQL Application Development (Java)

Couchbase 5.x (5.5) / Couchbase Java SDK 2.x+ (2.6.1)

Lab Workbook

Leo Schuman and Tony Piazza
August 20, 2018

Lab 1 - Install Couchbase server, project files, and REST API tool for testing

Objectives

A	Install and configure Couchbase Server 5.x as a single node
B	Survey the Couchbase administration console
C	Import Java projects into Eclipse
D	Install REST API tool

This lab assumes you do <u>not</u> have Couchbase Server EE 5.x installed. If you do, you should be able to complete all labs in this course using your existing installation. But, you may encounter small differences relative to the lab descriptions, depending on your version and configuration.

This course assumes you are already familiar with, and will use the open source Eclipse IDE, configured for Java 1.8+ development, with the M2Eclipse Maven tools installed. The project files can also be imported into other IDE's, such as IntelliJ IDEA.

A. Install and configure Couchbase Server 5.x as a single node cluster

1. Download *Couchbase Server Enterprise Edition 5.x* for your operating system.

   http://www.couchbase.com/downloads

2. Review the Release Notes, and install as described in the documentation for your OS:

   http://docs.couchbase.com

   Note, on Windows, you must run the installer using elevated Administrator permissions. To explore container-based installation, see:

   http://www.couchbase.com/containers.

3. Unless the Couchbase Setup tool launches automatically, open a web browser, and browse to this URL to launch the Setup tool.

   http://localhost:8091

4. Select *Setup New Cluster*

5. In the Setup tool, review all settings and accept all defaults, <u>except</u> for changing these:

    ❏ Cluster Name: *Training Cluster*
    ❏ Admin Username: *Administrator*
    ❏ Password: *password*
    ❏ Accept the Terms and Conditions (*check*)
    ❏ Configure Disk, Memory, Services (*click*)
        ❏ Data Service (*check*): *2048mb*
        ❏ GSI Service (*check*): *512mb*
        ❏ FTS Service (*check*): *512mb*
        ❏ Query Service (*check*)
        ❏ Memory-Optimized Global Secondary Indexes (*check*)

    Note, if requested by your local firewall, accept incoming network connections for *beam.smp*, *memcached*, *epmd*, *indexer*, *moxi*, *projector*, *cbq-engine*, and *cbft*. A full list of port requirements is available here:

    https://developer.couchbase.com/documentation/server/current/install/install-ports.html

    Note, the settings above run a single low-impact Couchbase instance on a local system for learning purposes <u>only</u>. Please see the Couchbase documentation for guidance on memory sizing for production machines, or for any performance-related purpose.

6. After completing Setup, you should see this screen. If not, browse http://localhost:8091.
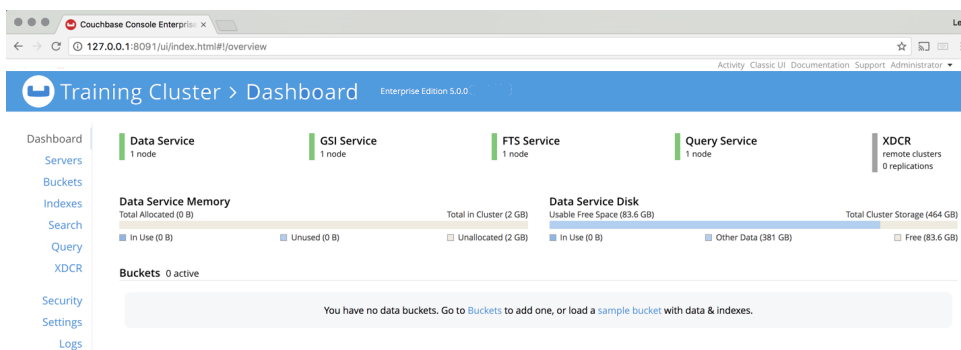
B. Survey the Couchbase administration console

7. In the Couchbase UI, navigate to and briefly review each top-level screen.

   For a solid introduction to the essential concepts of Couchbase technology, please take this free online training at http://training.couchbase.com/online.

   CB030 - Essentials of Couchbase NoSQL Technology

   Documentation is available at http://docs.couchbase.com
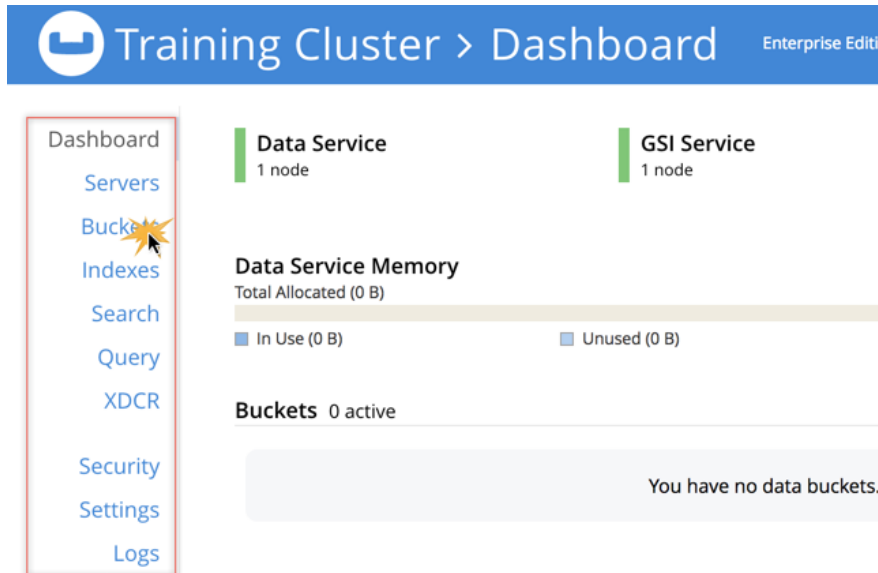
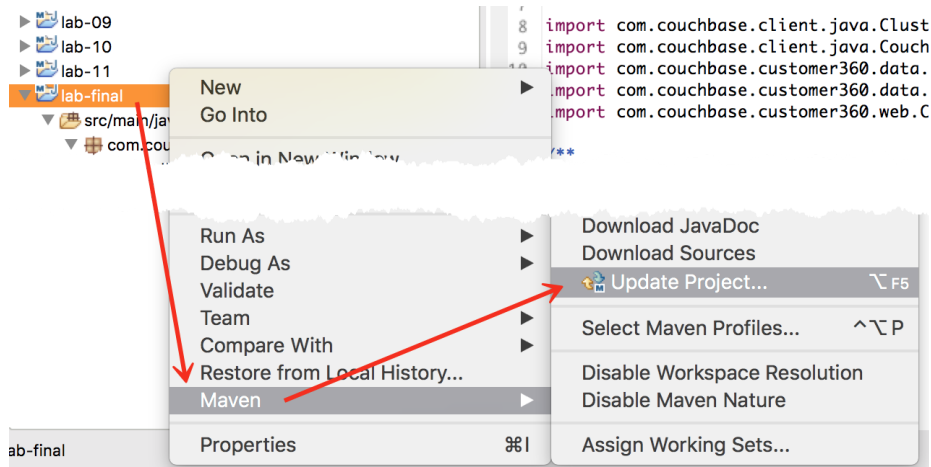### C. Import Java projects into Eclipse

5. From the learning management system, download the Java projects archive for these labs to your desktop:

   CB130J-Eclipse-Projects-[version].zip

6. Launch Eclipse and import all projects in this archive. After import, you should see this:
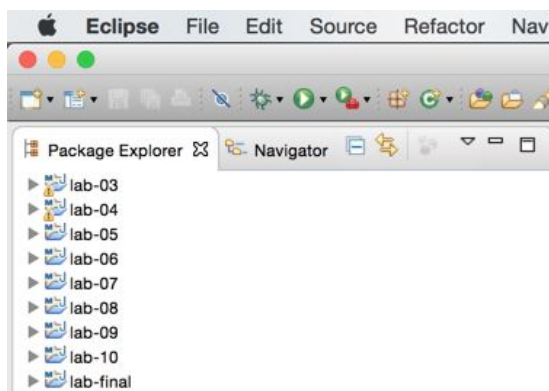
7. Update each Maven project, to ensure local JAR file availability.
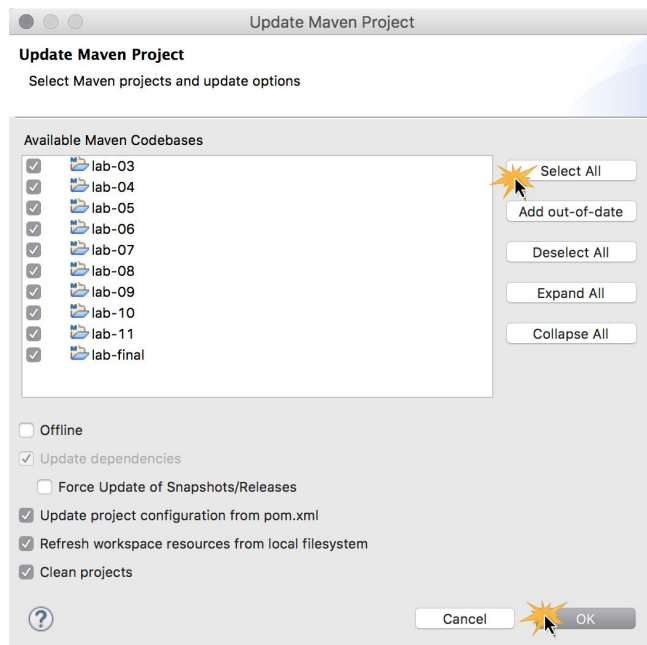
*D. Install HTTP Tool in Firefox browser*

> **Note, a plugin named HTTP Tool will be used in the video lessons, but you may use any REST API testing tool you prefer. <u>You might prefer to use Postman</u>, instead of installing HTTP Tool yourself.**
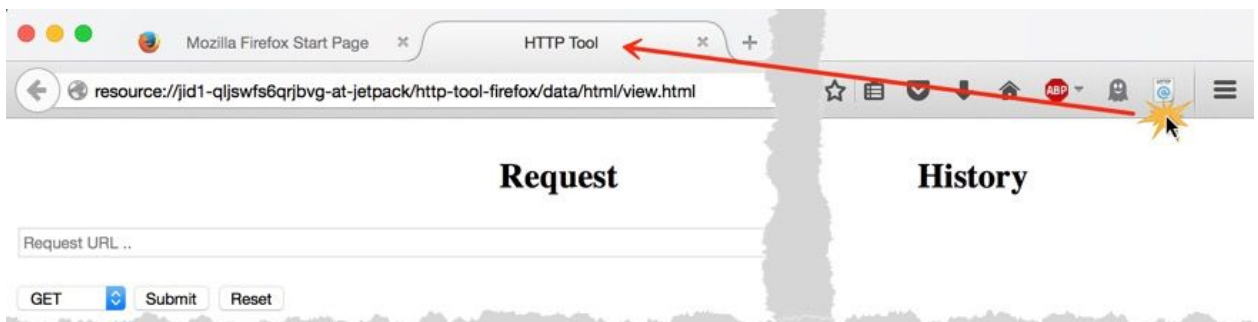>
> https://www.getpostman.com/

8. In a Firefox web browser, open the Add-ons Manager.
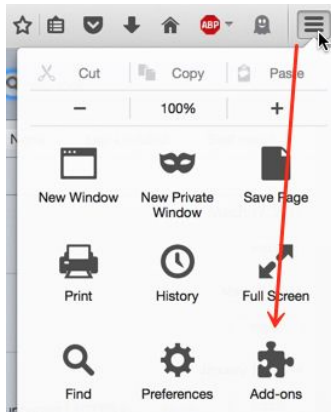
9. Search for and install HTTP Tool, for use testing the REST API to be created ahead.
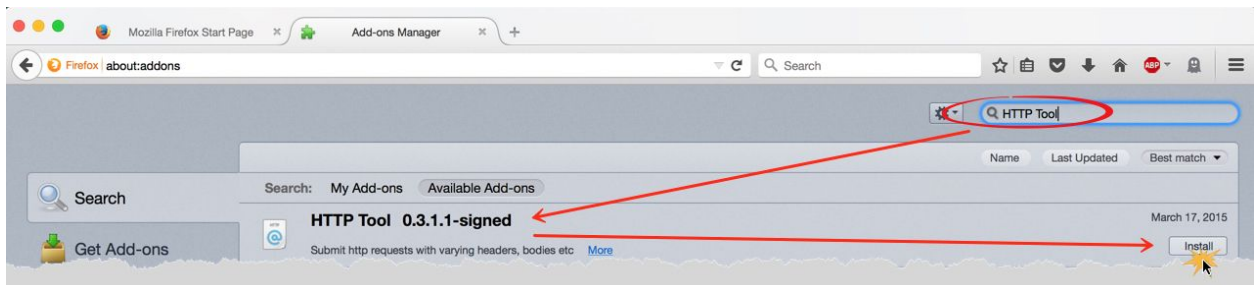
10. Launch HTTP Tool to verify installation.

End of Lab

Lab 2 - Manually create a Couchbase data bucket and JSON document, then preview the final application

Objectives

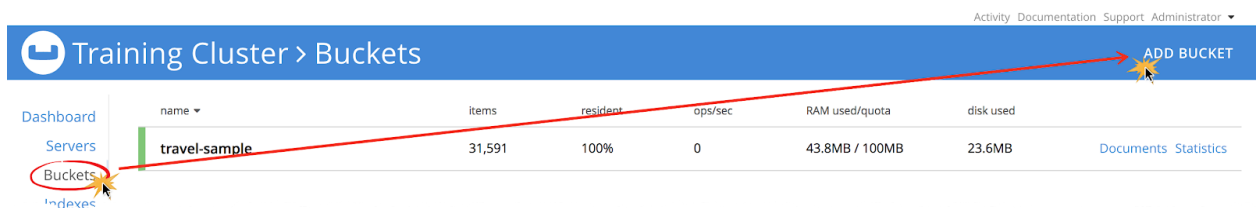| A	Manually create a data bucket
B	Manually add a document to a data bucket
C	Configure user security to enable customer360 bucket access
D	Run the final application and preview its behavior
E	Load additional documents into customer360 using cbimport

In this lab, you configure the data bucket and document you will use in later labs. Then, you run the final application to observe some of the behaviors you'll be creating ahead.

Note, throughout this course, code and JSON which <u>you</u> must type in appears in blue.

A. Manually create a data bucket

1. In the Couchbase administration console, navigate to the *Buckets* view, and select *Add Bucket*. Review all settings, and accept all defaults, except for these:
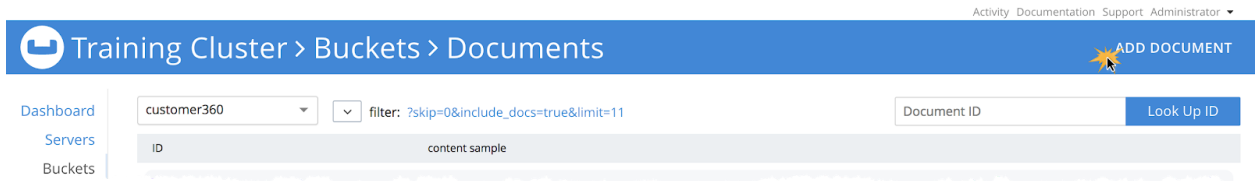
    ❏ Name: *customer360*
    ❏ Memory Quota: *1024mb*
    ❏ Advanced Settings (*click*)
        ❏ Replicas: *Disable (uncheck)*
        ❏ Flush: *Enable (check)*

B. Manually add a document to a data bucket

2. When the bucket is available, open its *Documents* view, and select *Add Document*.
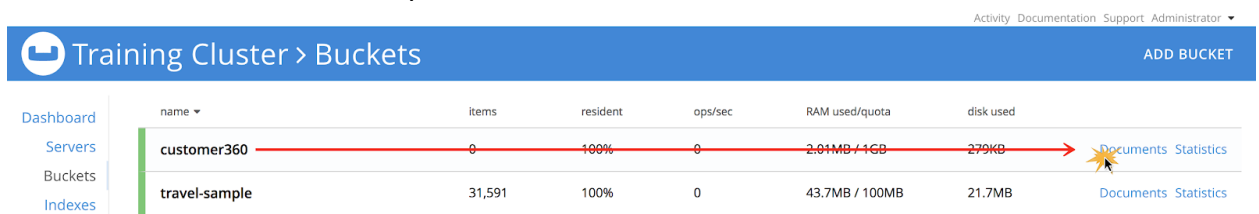
3. Assign the following value as the Document ID, then create the document:

   `customer::bblue22`

   Note, the general format *type::id* is a common convention for Document IDs in Couchbase, and is used in this course. But, it is only a convention. It is not required.

4. Edit and save the document with the following JSON value:

```
{
  "email": "bblue22@mailinator.com",
  "userName": "bblue22",
  "firstName": "Betty",
  "lastName": "Blue"
}
```
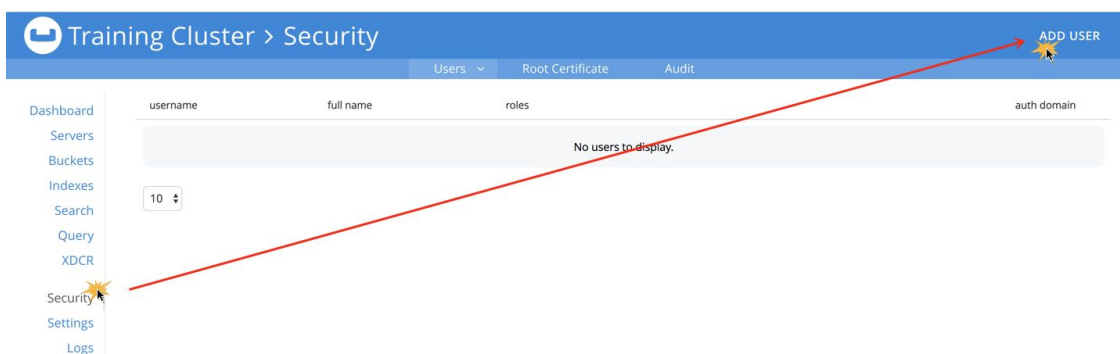
   Note, you can copy and paste all JSON snippets from this document.

5. In the navigation trail, click *Documents* to view the summary list of documents in the *customer360* bucket
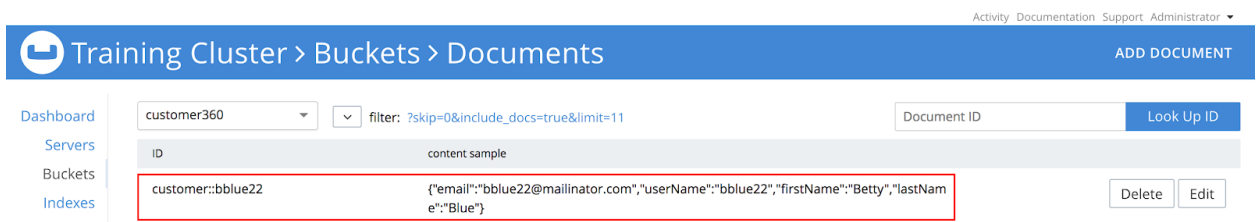
C. Configure user security to enable customer360 bucket access

6. In the Couchbase UI, select the Security tab to open User configuration.

7. Add a new user matching the name of the bucket created above (e.g., "*customer360*"), set a password (e.g., "*password*"), and configure this user for Full Access to this bucket.

Note, unless otherwise specified in the client, a SDK request is mapped to a User account matching the name of the Bucket against which the request is made. This account will provide access to otherwise anonymous clients when making requests to the *Customer360* bucket, provided they send the password specified for this account.

Much more granularly identified and secured approaches are also available. See the Roles Based Access Control (RBAC) sections of the Couchbase Server 5.0 Enterprise Edition documentation for further detail.

8. Save your changes to this new account. You should see the following.

D. Run the final application and preview its behavior

6. In Eclipse, close projects which are unrelated to the *final* project.
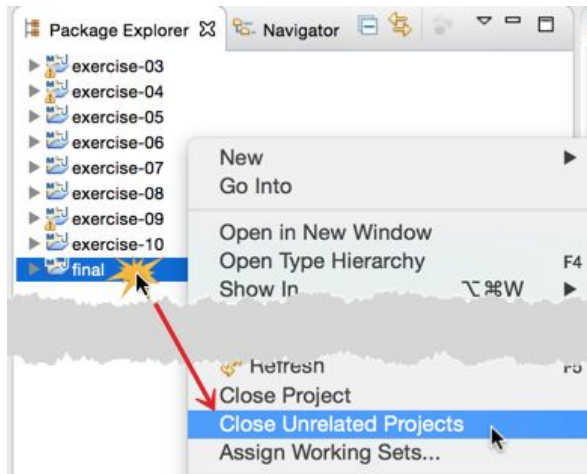
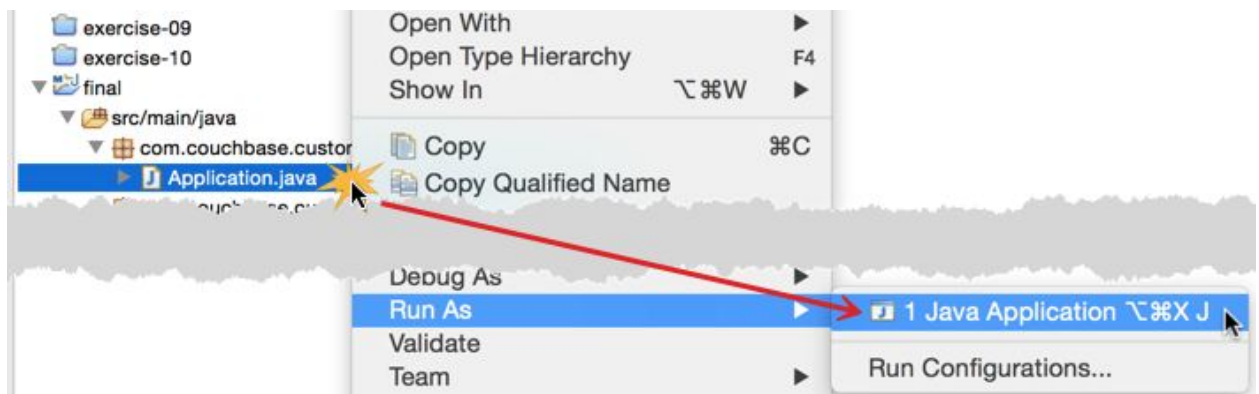7. In the *final* project, open *Application.java*, right-click and run it as a Java application.

8. If your Couchbase server is running, and you have created the *customer360* bucket, you should see the following:

Note, port 4567 is in use by SparkJava, which is being used to create this REST API. The Couchbase SDK itself connects to Couchbase Server via ports 11210, 11211, etc.

9. In Firefox, launch HTTP Tool, and make this GET request to retrieve the document:

❏ HTTP Verb (Method): GET
❏ URL for SparkJava: *http://localhost:4567*
❏ Base application URL: */customer360*
❏ Controller extension: */customer*
❏ Document ID: *customer::bblue22*

`http://localhost:4567/customer360/customer/customer::bblue22`

Request

```
http://localhost:4567/customer360/customer/customer::bblue22
```
GET ⬍   Submit   Reset

| Headers | Body content |

| **Name** | **Value** | + |

Response

| Headers | Body content | Status (200) |

Copy to clipboard

```
{
    "created": "2017-09-26T15:15:52-0700",
    "updated": "2017-09-26T15:15:52-0700",
    "cas": 1506462643411878000,
    "email": "bblue22@mailinator.com",
    "userName": "bblue22",
    "firstName": "Betty",
    "lastName": "Blue",
    "type": "com.couchbase.customer360.domain.Customer"
}
```

Note, port 4567 is in use by SparkJava, which is being used to create this REST API. The Couchbase SDK itself connects to Couchbase Server via ports 11210, 11211, etc.

10. Stop the running Java application.

E. Load additional documents into customer360 using cbimport

You want to bulk load data into the bucket you have created.

11. Download the *customer360-data.json* file to your desktop or similar location. Using Eclipse or your preferred tool, examine the structure of this document.

```
customer360-data.json  ×
1  {"username": "aahedelides4347", "updated": "2015-07-24T15:59:26", "firstName": "Ellen", "created": "2015-03-
2  {"username": "aahedotes2079", "updated": "2015-07-24T15:49:09", "firstName": "Valentin", "created": "2014-09-
3  {"username": "aahingumpire656", "updated": "2015-07-24T15:45:04", "firstName": "Billie", "created": "2015-02
4  {"username": "abatescagy1787", "updated": "2015-07-24T15:48:25", "firstName": "Maria", "created": "2015-02-2
5  {"username": "abbeyscoasts4036", "updated": "2015-07-24T15:57:13", "firstName": "Anton", "created": "2014-08-
6  {"username": "abedchurns866", "updated": "2015-07-24T15:45:58", "firstName": "Peter", "created": "2014-10-11
7  {"username": "abetsigloos748", "updated": "2015-07-24T15:45:30", "firstName": "Gene", "created": "2015-01-09
```

Note, each line of *customer360-data.json* contains a JSON document (object). Each document includes a unique *id* value.

```
customer-sample.json  ×
1   {
2      "username": "aahedelides4347",
3      "updated": "2015-07-24T15:59:26",
4      "firstName": "Ellen",
5      "created": "2015-03-10T04:33:14",
6      "lastName": "Wuori",
7      "email": "ellen.wuori67@games.com",
8      "billingAddress": {
9         "postalCode": "99632",
10        "country": "FI",
11        "state": null,
12        "line1": "4028 mannerheimintie",
13        "city": "kolari"
14     },
15     "type": "customer",
16     "id": "aahedelides4347"
17  }
```

12. Open a Terminal (Command) window, navigate to the Couchbase *bin* folder, and briefly review the contents of this folder. Notice the *cbimport* and *cbdocloader* tools.

macOS	/Applications/Couchbase Server.app/Contents/Resources/couchbase-core/bin/
Windows	C:\Program Files\Couchbase\Server\bin\
Linux	/opt/couchbase/bin/

```
● ● ●                    📁 bin — -bash — 109×24
[couchbase:CB130p Schuman$ cd /Applications/Couchbase\ Server.app/Contents/Resources/couchbase-core/bin/    ]
[couchbase:bin Schuman$ ls                                                                                   ]
cbbackupmgr             cbworkloadgen           generate_cert
cbbrowse_logs           couch_compact           gometa
cbcollect_info          couch_dbck              goport
chcompact               couch_dbdump            gosecrets
cbdocloader             couch_dbinfo            goxdcr
cbdump-config           couch_view_file_merger  gozip
cbenable_core_dumps.sh  couch_view_group_cleanup  indexer
cbepctl                 couch_view_group_compactor  install
cbexport                couch_view_index_builder  jeprof
cbft                    couch_view_index_updater  mcctl
cbft-bleve              couchbase-cli           mclogsplit
cbimport                couchbase-server        mcstat
cbindex                 couchdb                 mctimings
cbindexperf             couchjs                 memcached
chindexplan             c            pl         Scope
```

13. Add this *bin* folder to the PATH environment variable for your operating system, so that its commands may be invoked from any command line location. See your operating system documentation for details on this process.

14. In the Terminal, navigate to the */CB130j* folder, which contains *customer360-data.json.*

15. Use *cbimport* to load the JSON documents in the *customer360-data.json* file to the *customer360* bucket. Assign the *id* element of each record as its document key, prefixed by "*customer*" as a document *type* identifier, using "*::*" as a separator.

   ❏ Cluster (-c): *couchbase://127.0.0.1*
   ❏ Username (-u): *Administrator*
   ❏ Password (-p): *password*
   ❏ Bucket (-b): *customer360*
   ❏ Format (-f): *lines*
   ❏ Dataset (-d): *file://customer360-data.json*
   ❏ Threads (-t): *2*
   ❏ Key Pattern to Generate (-g): *customer::%id%*

   cbimport json -c couchbase://127.0.0.1 -u Administrator -p password -b customer360 -f lines -d file://customer360-data.json -t 2 -g customer::%id%

   For full detail on using *cbimport*, see the documentation:

   https://developer.couchbase.com/documentation/server/current/tools/cbimport.html

   Note, any document value can be used for key generation. It is a common JSON design pattern to prefix document keys with a type identifier (e.g., "*customer*::aaa123"). When using *cbimport*, prefixing can be specified using key generation, as shown above. When using *cbdocloader* and the prescribed Couchbase sample file format, the file names of your documents, which may include prefixing as shown in the provided archive, become key names for the loaded documents.

   For an introduction to data modeling in JSON, please take this free Online Training course at http://training.couchbase.com/online.

   CB105 - Introduction to Data Modeling in JSON

16. In the Couchbase UI, verify you've loaded 5,000 additional documents to the *customer360* bucket, and open the *Documents* screen.

17. In the *Documents* screen, Edit the first document to review its structure.

ID	content sample		
customer::aahedelides4347	{"username":"aahedelides4347","updated":"2015-07-24T15:59:26","firstName":"Ellen","cre-ated":"2015-03-10T04:33:14","lastName":"Wuori","email":"ellen.wuori67@games.com","billingAddress": {"postalCode":"9	Delete	Edit

Customer360 filter: ?skip=0&include_docs=true&limit=6   Document ID   Look Up ID

18. Notice the related metadata object.

customer::aahedelides4347

```
1  {
2    "username": "aahedelides4347",
3    "updated": "2015-07-24T15:59:26",
4    "firstName": "Ellen",
5    "created": "2015-03-10T04:33:14",
6    "lastName": "Wuori",
7    "email": "ellen.wuori67@games.com",
8    "billingAddress": {
9      "postalCode": "99632",
10     "country": "FI",
11     "state": null,
12     "line1": "4028 mannerheimintie",
13     "city": "kolari"
14   },
15   "type": "customer",
16   "id": "aahedelides4347"
17 }
```

Delete   Save As...   Save

```
1  {
2    "id": "customer::aahedelides4347",
3    "rev": "1-
     14b9180d2d2b00000000000002000006",
4    "expiration": 0,
5    "flags": 33554438
6  }
```

End of Lab

# **Lab 3** - Create a Couchbase cluster reference and open a specified bucket

Objectives

A	Survey the *customer360* application
B	Understand how the cluster reference is created
C	Use the cluster to open a bucket reference, with and without password

In this lab, you survey and orient yourself to the application code. Then, you create a cluster reference, and write code to open the bucket you created in the prior lab, within this cluster, using a password <u>if</u> the bucket has been secured.

A. Survey the customer360 application

1. In Eclipse, open the the *lab-03* project, and close unrelated projects.
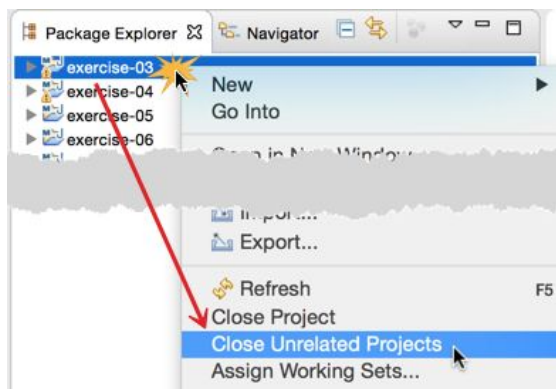
2. Review the code in this project, in light of the descriptions provided in the related video materials. Focus on reviewing these three classes. You may also wish to review the code in the *final* project.

   - *Application.java*
   - *BaseController.java*
   - *CouchbaseRepository.java*

   Note, this application describes a straightforward way to build a REST-based interface for Couchbase, relying on the Sinatra-inspired Spark web framework, and synchronous API calls. Asynchronous, reactive approaches using RxJava, are taught in live, instructor-led Couchbase courses, and the documentation.

   To maximize clarity for learners who may be new to Java, exception handling is minimized in the labs, though is fully implemented in the lab-final project.

B. Understand how the cluster reference is created

3. In *Application.java*, review the *main* method. Specifically, notice:

   ❏ How the node addresses are loaded from a properties file
   ❏ How a cluster reference is created for those node(s)
   ❏ How the repository is created for this cluster for the *customer360* bucket
   ❏ How the cluster is used to create the controllers

```java
public static void main(String[] args) {
        final Properties props = System.getProperties();
        try {
                props.load(getSystemResourceAsStream(PROPERTIES_FILENAME));
        } catch (IOException e) {
                System.err.println("Unable to open " + PROPERTIES_FILENAME);
        }
        String[] nodes =
                props.getProperty(NODES_KEY, NODES_DEFAULT_VALUE).split(",");
        Cluster cluster = CouchbaseCluster.create(nodes);
        Repository repo =
                new CouchbaseRepository(cluster, "customer360");

        new CustomerController(BASE_URL, repo);
        new ProductController(BASE_URL, repo);
        new InteractionController(BASE_URL, repo);
}
```

C. Use the cluster to open a bucket reference, with and without password

4. In *CouchbaseRepository.java*, notice the *bucket* property.

```java
public class CouchbaseRepository implements Repository {
        private final JsonConverter converter = new JacksonConverter();
        private final JsonTranscoder transcoder = new JsonTranscoder();
        private Bucket bucket;
```

5. In the first constructor, delete the *TODO* and thrown *UnsupportedOperationException*.
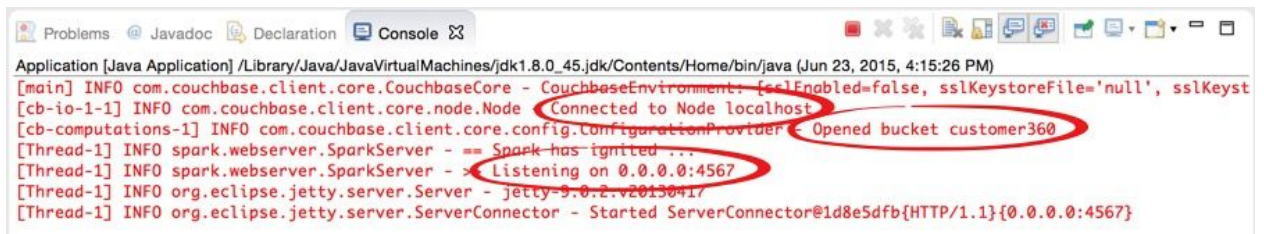6. Use the *cluster* reference to open the passed bucket name.

```java
public CouchbaseRepository(Cluster cluster, String bucketName) {
        bucket = cluster.openBucket(bucketName);
}
```

7. In the second constructor, which supports a password, again open *bucketName*, but this time also using the *bucketPassword*.

```java
public CouchbaseRepository(Cluster cluster, String bucketName,
        String bucketPassword) {
        bucket = cluster.openBucket(bucketName, bucketPassword);
}
```

Manually test the application

8. Run the *lab-03* project. You should see this console output, indicating you have successfully opened the bucket you've created.

10. Stop the running application.

End of Lab

Lab 4 - Get, serialize, deserialize, and return JSON from Couchbase

Objectives

A	Retrieve a document by its key
B	Deserialize a Java type to a JSON string
C	(Optional) Review REST method mapping in the Spark framework

In this lab you get a document by its key. More importantly, though, you begin learning how to handle the various states through which JSON data may pass in an application:

> *String*
> Application-specific Java domain type (e.g., *Customer*, *Product*, etc.)
> *JsonObject*
> *JsonDocument*

A. Retrieve a document by its key

1. In Eclipse, open the *lab-04* project, and close unrelated projects.
2. In *CouchbaseRepository.java*, locate the *findById* method, review its signature, and delete the TODO and placeholder exception.
3. Declare a local *JsonDocument* variable named *doc*.
4. Use the *get* method of the *bucket* reference to retrieve the passed Document ID, and assign the returned value to *doc*.

```
public <T extends Entity> T findById(String id, Class<? extends T> type) {
    JsonDocument doc = bucket.get(id);
}
```

B. Deserialize a Java type to a JSON string

5. In *CouchbaseRepository.java*, above the constructors, locate the *converter* and *transcoder* class variable declarations, and consider them in light of the descriptions provided in the related video materials. Specifically, notice:

> ❏ *converter* could be any JSON converter which maps JSON strings to Java types
> ❏ *converter* maps JSON strings to Java domain objects of a specified type
> ❏ *transcoder* converts JSON strings to the Couchbase *JsonObject* type

```
public class CouchbaseRepository implements Repository {
    private final JsonConverter converter = new JacksonConverter();
    private final JsonTranscoder transcoder = new JsonTranscoder();
    private Bucket bucket;
```

6. Locate the *fromJsonDocument* method, and review it in light of the descriptions provided in the related video materials. Specifically, notice:

   ❏ This application uses generics to flexibly handle multiple Java domain types, all of which derive from a base type called *Entity*
   ❏ The Bucket API gets a *JsonDocument* object, holding its *JsonObject* as *content*
   ❏ Calling *toString* on a *JsonObject* returns the underlying string of JSON
   ❏ *JacksonConverter* converts a JSON string to a specified Java type

```java
protected <T extends Entity> T fromJsonDocument(JsonDocument doc,
  Class<T> type) {
      if (doc == null) {
            throw new IllegalArgumentException("document is null");
      }
      JsonObject content = doc.content();
      if (content == null) {
            throw new IllegalStateException("document has no content");
      }
      if (type == null) {
            throw new IllegalArgumentException("type is null");
      }
      T result = converter.fromJson(content.toString(), type);
      return result;
}
```

7. In the *findById* method, if the retrieved document is not *null*, add code to convert it to the specified Java *type* using the *fromJsonDocument* method, and *return* it.

```java
public <T extends Entity> T findById(String id, Class<? extends T> type) {
      JsonDocument doc = bucket.get(id);
      return doc == null ? null : fromJsonDocument(doc, type);
}
```

C. (Optional) Review REST method mapping in the Spark framework

8. (Optional) In *BaseController.java*, review how GET requests are mapped to retrieve the ID on the URL, and pass it to the *findById* method.
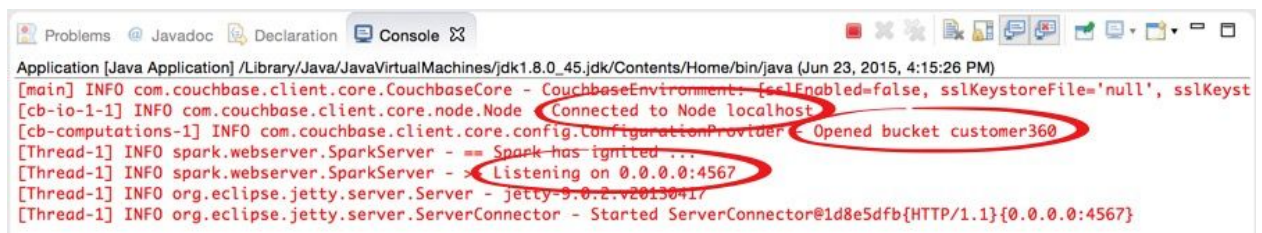
```java
if(!methodList.contains(HttpMethod.get)) {
  Spark.get(baseURL + "/:id", new Route() {
    @Override
    public Object handle(Request request, Response response) {
      T entity = repo.findById(request.params(":id"), type);
      if(entity == null) {
        return error(response, HttpStatus.NOT_FOUND_404);
      } else {
        contentLocation(response, baseURL + "/" + entity.getId());
        return ok(response, entity);
      }
    }
  });
}
```

9. (Optional) In *BaseController.java*, review how the *ok(Response, T)* method uses the *converter* to deserialize the Java domain type back to a JSON string.

```java
protected <T> String ok(Response response, T entity) {
        response.type("application/json");
        response.status(HttpStatus.OK_200);
        return converter.toJson(entity);
}
```

Manually test the application

10. Run the *lab-04* project. You should see this console output:

9. In Firefox, launch HTTP Tool, and make this GET request to retrieve the document:

- ❏ HTTP Verb (Method): GET
- ❏ URL for SparkJava: *http://localhost:4567*
- ❏ Base application URL: */customer360*
- ❏ Controller extension: */customer*
- ❏ Document ID: *customer::bblue22*

```
http://localhost:4567/customer360/customer/customer::bblue22
```

Request

http://localhost:4567/customer360/customer/customer::bblue22

GET   Submit   Reset
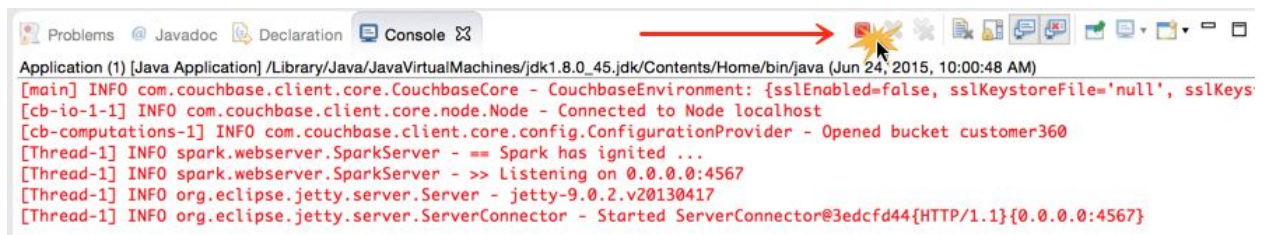
| Headers | Body content |

Response

| Headers | Body content | Status (200) |

Copy to clipboard

```
{
    "created": "2015-07-01T08:39:52-0700",
    "updated": "2015-07-01T08:39:52-0700",
    "cas": 262894632108032,
    "email": "bblue22@mailinator.com",
    "userName": "bblue22",
    "firstName": "Betty",
    "lastName": "Blue",
    "type": "com.couchbase.customer360.domain.Customer"
}
```

10. Stop the running application.


```
Problems  @ Javadoc  Declaration  Console ⊠

Application (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Home/bin/java (Jun 24, 2015, 10:00:48 AM)
[main] INFO com.couchbase.client.core.CouchbaseCore - CouchbaseEnvironment: {sslEnabled=false, sslKeystoreFile='null', sslKeys
[cb-io-1-1] INFO com.couchbase.client.core.node.Node - Connected to Node localhost
[cb-computations-1] INFO com.couchbase.client.core.config.ConfigurationProvider - Opened bucket customer360
[Thread-1] INFO spark.webserver.SparkServer - == Spark has ignited ...
[Thread-1] INFO spark.webserver.SparkServer - >> Listening on 0.0.0.0:4567
[Thread-1] INFO org.eclipse.jetty.server.Server - jetty-9.0.2.v20130417
[Thread-1] INFO org.eclipse.jetty.server.ServerConnector - Started ServerConnector@3edcfd44{HTTP/1.1}{0.0.0.0:4567}
```

End of Lab

Lab 5 - Serialize, transcode, and insert a JSON document to Couchbase

Objectives

A	Serialize JSON to a Java domain type
B	Transcode a Java domain type to a JsonDocument
C	Insert a new JSON document
D	Transcode a JsonDocument to a Java domain type

In this lab, you handle string input passed via the REST API, by serializing and transcoding as needed so that it is both available within the application as a Java domain type, as well as converted to a *JsonDocument* and inserted into Couchbase.

A. Serialize JSON to a Java domain type

1. In Eclipse, open the *lab-05* project, and close unrelated projects.
2. In *BaseController.java*, review how POST requests are handled. Specifically, notice:

   ❏ The request body is converted from JSON to a Java domain object of its type
   ❏ The Java domain object and its type pass to *CouchbaseRepository.create()*
   ❏ The *create* result is passed to the *ok* method to serialize and return the response

```
if(!methodList.contains(HttpMethod.post)) {
    Spark.post(baseURL, new Route() {
        @Override
        public Object handle(Request request, Response response) {
            T entity = converter.fromJson(request.body(), type);
            contentLocation(response, baseURL + "/" + entity.getId());
            T result = repo.create(entity, type);
            return ok(response, result);
        }
    });
}
```

B. Transcode a Java domain type to a JsonDocument

3. In *CouchbaseRepository.java*, locate the *create* method, review its signature, and delete the *TODO* and placeholder exception. Notice the method receives a Java domain object, named *entity*, along with a reference to its *type*.

```
public <T extends Entity> T create(T entity, Class<? extends T> type) {

}
```

4. In the *create* method, pass the Java domain object passed in as *entity* to the *toJsonDocument* method for conversion, and assign the result to a *JsonDocument* variable named *docIn*.

```java
public <T extends Entity> T create(T entity, Class<? extends T> type) {
    JsonDocument docIn = toJsonDocument(entity);

}
```

5. (Optional) Locate the *toJsonDocument* method, and review it in light of the descriptions provided in the related video materials. Specifically, notice:

   ❏ This method converts a Java object, with some ID value, to a *JsonDocument*
   ❏ The *converter* deserializes a Java object into a JSON string
   ❏ The *transcoder* serializes a JSON string into a *JsonObject*
   ❏ *JsonDocument.create()* builds a *JsonDocument* from an ID value and *JsonObject*

```java
protected <T extends Entity> JsonDocument toJsonDocument(T source) {
    if (source == null) {
        throw new IllegalArgumentException("entity is null");
    }
    String id = source.getId();
    if (id == null) {
        throw new IllegalStateException("entity ID is null");
    }
    try {
        JsonObject content =
            transcoder.stringToJsonObject(converter.toJson(source));
        JsonDocument doc = JsonDocument.create(id, content);
        return doc;
    } catch (Exception e) {
        throw new RepositoryException(e);
    }
}
```

C. Insert a new JSON document

6. In the *create* method, use the *insert* method of the *bucket* to pass the new document to Couchbase for insertion. Assign the result of this method - which will include the inserted document's time stamp and CAS value - to a *JsonDocument* named *docOut*.

```java
public <T extends Entity> T create(T entity, Class<? extends T> type) {
    JsonDocument docIn = toJsonDocument(entity);
    JsonDocument docOut = bucket.insert(docIn);

}
```

D. Transcode a JsonDocument to a Java domain type

7. Convert the resulting *JsonDocument* back into a Java domain object of its correct type, using the *fromJsonDocument* method, and *return* the result.

```java
public <T extends Entity> T create(T entity, Class<? extends T> type) {
        JsonDocument docIn = toJsonDocument(entity);
        JsonDocument docOut = bucket.insert(docIn);
        return fromJsonDocument(docOut, type);
}
```

Manually test the application

8. Run the *lab-05* project. You should see this console output:
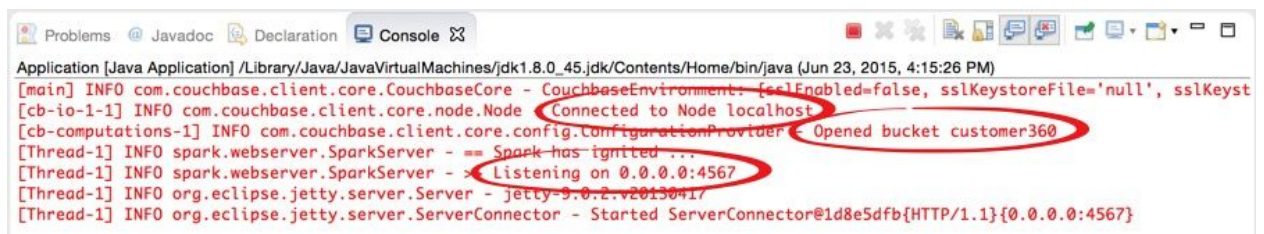
9. In Firefox, launch HTTP Tool, and make this POST request to insert a new document:

- ❏ HTTP Verb (Method): POST
- ❏ URL for SparkJava: *http://localhost:4567*
- ❏ Base application URL: */customer360*
- ❏ Controller extension: */customer*
- ❏ Document ID: *none for a POST (must be in body)*

```
http://localhost:4567/customer360/customer
```

Body Content (note, you can copy and paste this code from this document):

```json
{
  "id": "customer::rred33",
  "email": "rickyred33@mailinator.com",
  "userName": "rred33",
  "firstName": "Ricky",
  "lastName": "Red",
  "billingAddress":
    {
      "line1": "1234 Five St.",
      "city": "Portland",
      "state": "OR",
      "postalCode": "97203"
    }
}
```

9. You should see the inserted document, with time stamp and CAS (not yet implemented at this point in the application, and so defaults to 0), in the Response Body Content.

Response

Headers	Body content	Status (200)

Copy to clipboard

```
{
    "id": "customer::rred33",
    "created": "2015-06-24T15:27:04-0700",
    "updated": "2015-06-24T15:27:04-0700",
    "cas": 0,
    "email": "rickyred33@mailinator.com",
    "userName": "rred33",
    "firstName": "Ricky",
    "lastName": "Red",
    "billingAddress": {
        "line1": "1234 Five St.",
        "city": "Portland",
        "state": "OR",
        "postalCode": "97203"
    },
    "type": "com.couchbase.customer360.domain.Customer"
}
```

10. Reload HTTP Tool in the browser to fully clear its state.

Note, personal browser configuration choices may affect how the HTTP Tool plugin behaves when switching among HTTP verbs. To reduce related issues, reload HTTP Tool in your browser before configuring each new test request.

11. Verify the document insertion by making a GET request for the new document. You should receive the same response as you saw returned from the insertion above.

- ❏ HTTP Verb (Method): GET
- ❏ URL for SparkJava: *http://localhost:4567*
- ❏ Base application URL: */customer360*
- ❏ Controller extension: */customer*
- ❏ Document ID: *customer::rred33*

http://localhost:4567/customer360/customer/**customer::rred33**

12. Stop the running application.

End of lab

Lab 6 - Index and query using N1QL

Objectives

A	Create a primary index and execute queries via cbq
B	Retrieve nested documents by either value or Document ID (key)
C	Implement a parametrized N1QL query through the Java SDK

In this lab, you use *cbq* to index the *customer360* data bucket for ad hoc queries, and then execute N1QL queries against this data bucket. You then refactor the *findById()* method implemented in the prior lab, using N1QL and the Couchbase *query()* API.

A. Create a primary index and execute ad hoc N1QL queries
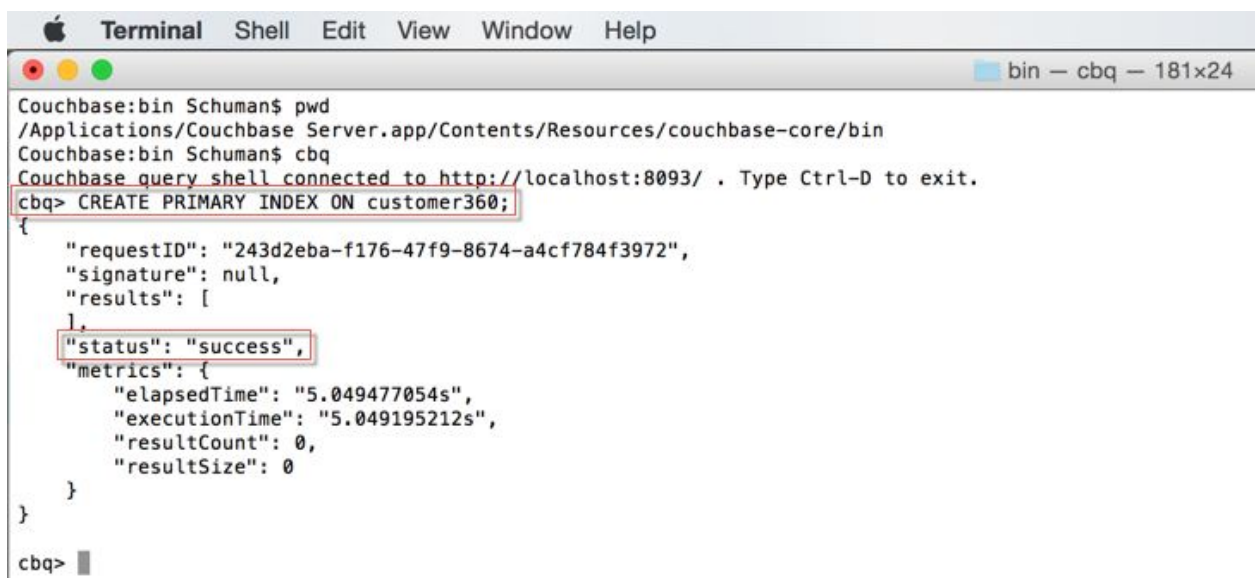
1. Open a terminal window and navigate to the *../Contents/Resources/couchbase-core/bin* folder of your local Couchbase instance.

2. Use the "cbq" command to launch *cbq* from the command line.
3. To enable ad hoc queries against the *customer360* data bucket, execute the following N1QL statement, and verify that it succeeds.

```
CREATE PRIMARY INDEX ON customer360;
```

B. Retrieve nested documents by either value or Document ID (key)

4. Select all keys and values for 10 documents from the *customer360* data bucket.

```
SELECT * FROM customer360 LIMIT 10;
```


Notice the top level element for <u>each</u> result is named for the data bucket: *customer360*.

5. Select the document created in the prior lab, by the *id* key which was written into the document based on the corresponding *id* property of the *Customer* domain object in *customer360*. (Note, Document ID values are not automatically embedded in documents by Couchbase, this behavior is pre-built as part of the *customer360* application).

```
SELECT * FROM customer360 WHERE id = "customer::rred33";
```


Notice the *executionTime* for this query.

6. Select the same document, but this time extracted from its surrounding *customer360* element, and explicitly by its key (Document ID), rather than an embedded value.

```
SELECT customer360.* FROM customer360 USE KEYS "customer::rred33";
```


Notice the *executionTime* for this query. While the actual value will vary based on your hardware and configuration, it should be significantly faster than the prior query.

Also, notice the document is no longer wrapped in an element identifying its source as the *customer360* data bucket, but is now just the underlying data document itself.

To learn <u>much</u> more about querying and indexing JSON data using N1QL, please take our free online course:

CB110 - Intro to N1QL for SQL People

You may also wish to take the related free online course:

CB105 - Intro to Data Modeling in JSON

C. Implement a parametrized N1QL query through the Couchbase SDK

7. In Eclipse, open the *lab-06* project, and close unrelated projects.
8. In *CouchbaseRepository.java*, locate the *findById* method, declare a *JsonDocument* variable named *doc*, and comment out the *get* method implemented in the prior lab.

```
public <T extends Entity> T findById(String id, Class<? extends T> type) {
  JsonDocument doc;
  try {
    // doc = bucket.get(id);

  } catch (CouchbaseException e) {
    throw new RepositoryException(e);
  }
  return doc == null ? null : fromJsonDocument(doc, type);
}
```

9. In the try block, declare a String *statement* with the second N1QL query used above in *cbq*, but replacing the key value with *$1* as a placeholder for a positional parameter.
10. Create an *empty JsonArray* named *values*, and *add* the *id* parameter being passed to this method to the array.
11. *Build* a *N1qlParams* object named *params*, and set its *consistency* property to *ScanConsistency.REQUEST_PLUS*, to ensure strong consistency for this query.
12. Create a *ParameterizedN1qlQuery* named *query*, using *statement*, *values*, and *params*.
13. Update your *import* statements as needed:

```
...
import com.couchbase.client.java.query.N1qlParams;
import com.couchbase.client.java.query.ParameterizedN1qlQuery;
import com.couchbase.client.java.query.consistency.ScanConsistency;
...

public <T extends Entity> T findById(String id, Class<? extends T> type) {
  JsonDocument doc;
  try {
    // doc = bucket.get(id);
    String statement = "SELECT customer360.* FROM customer360 USE KEYS $1";
    JsonArray values = JsonArray.empty().add(id);
    N1qlParams params =
      N1qlParams.build().consistency(ScanConsistency.REQUEST_PLUS);
    ParameterizedN1qlQuery query =
      ParameterizedN1qlQuery.parameterized(statement, values, params);


  } catch (CouchbaseException e) {
    throw new RepositoryException(e);
  }
  return doc == null ? null : fromJsonDocument(doc, type);
}
```

14. Pass the *query* to the *query* method of the *bucket*, and assign the value returned to a *N1qlQueryResult* named *result*.
15. Assign *allRows* of of the *result* to a *List* of *N1qlQueryRow* values, named *list*.
16. If there are 0 *rows* in the *list*, set *doc* to *null*.
17. Else, *get* the first element in *list*, and assign it to a *N1qlQueryRow* named *firstRow*.
18. Assign the *value* of *firstRow* to a *JsonObject* named *firstRowObject*.
19. Use *firstRowObject* and the *id* passed to this method, to *create* a *JsonDocument*, assigning the result to *doc*.
20. Update your import statements as needed.

```java
...
import java.util.List;
import com.couchbase.client.java.query.N1qlQueryResult;
import com.couchbase.client.java.query.N1qlQueryRow;
...
public <T extends Entity> T findById(String id, Class<? extends T> type) {
  JsonDocument doc;
  try {
    // doc = bucket.get(id);
    String statement = "SELECT customer360.* FROM customer360 USE KEYS $1";
    JsonArray values = JsonArray.empty().add(id);
    N1qlParams params =
      N1qlParams.build().consistency(ScanConsistency.REQUEST_PLUS);
    ParameterizedN1qlQuery query =
      ParameterizedN1qlQuery.parameterized(statement, values, params);
    N1qlQueryResult result = bucket.query(query);
    List<N1qlQueryRow> list = result.allRows();
    if (list.size() == 0) {
      doc = null;
    } else {
      N1qlQueryRow firstRow = list.get(0);
      JsonObject firstRowObject = firstRow.value();
      doc = JsonDocument.create(id, firstRowObject);
    }


  } catch (CouchbaseException e) {
    throw new RepositoryException(e);
  }
  return doc == null ? null : fromJsonDocument(doc, type);
}
```

Note, from this point, having created a *JsonDocument* from the value returned for this Document ID, the *customer360* application should behave just as when using *get()*.

Note, this lab demonstrates fundamental aspects of the N1QL API, in context of this sample application. N1QL would primarily be used for selecting document sets, or querying by value, not getting single documents by ID, as shown.

17. Run the *lab-06* project. You should see this console output:

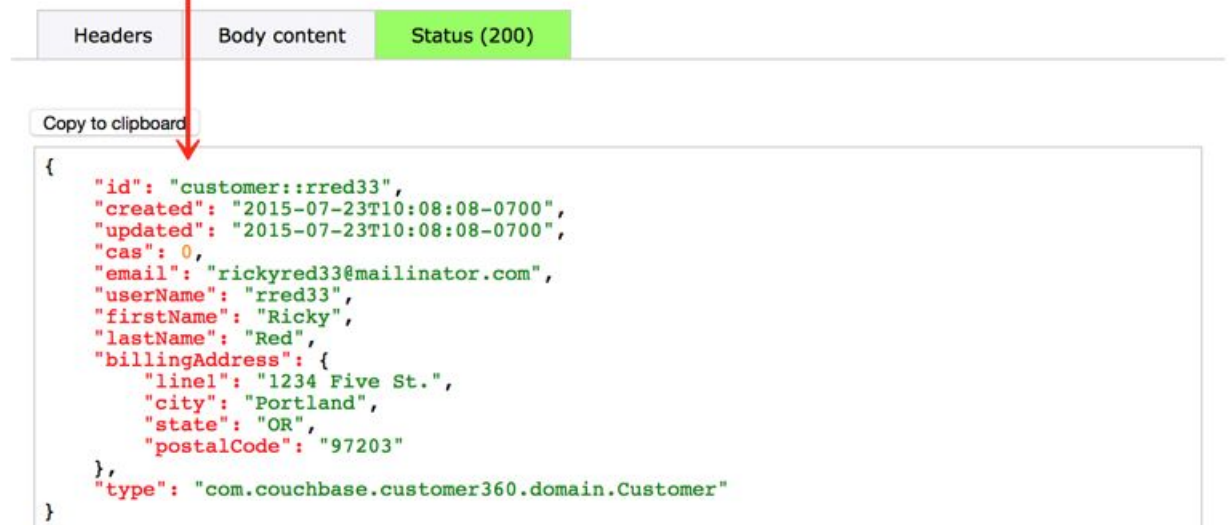18. In Firefox, launch HTTP Tool, and make this GET request to retrieve the document:

- ❏ HTTP Verb (Method): GET
- ❏ URL for SparkJava: http://localhost:4567
- ❏ Base application URL: /customer360
- ❏ Controller extension: /customer
- ❏ Document ID: customer::rred33

http://localhost:4567/customer360/customer/**customer::rred33**

19. Stop the running application.

End of Lab

Lab 7 - Replace ("update") a document

Objectives

A	Replace ("update") an existing document
B	Consider Couchbase write behavior

In this lab, you use the bucket's *replace* method to add a document which happens to have the same ID as an existing document. This effectively "updates" the prior version of the document.

In this scenario, the SDK returns an error on attempts to replace a document which does not already exist. Consider how this relates to the way Couchbase accepts incoming writes, particularly in comparison to the subsequent lab.

A. Replace ("update") an existing document

1. In Eclipse, open the *lab-07* project, and close unrelated projects.
2. In *CouchbaseRepository.java*, locate the *update* method, review its signature, and delete the *TODO* and placeholder exception. Notice the method receives a Java domain object, named *entity*, along with a reference to its *type*.

```
public <T extends Entity> T update(T entity, Class<? extends T> type) {

}
```

3. Use the *toJsonDocument* method to convert the entity passed to this method into a *JsonDocument*, named *docIn*.

```
public <T extends Entity> T update(T entity, Class<? extends T> type) {
        JsonDocument docIn = toJsonDocument(entity);

}
```

4. (Optional) Review the *toJsonDocument* method, to follow how JSON formatted data is flowing through this application.
5. Pass *docIn* to the *update* method of the *bucket*, and assign the result to a *JsonDocument* named *docOut*.

```
public <T extends Entity> T update(T entity, Class<? extends T> type) {
        JsonDocument docIn = toJsonDocument(entity);
        JsonDocument docOut = bucket.replace(docIn);

}
```

6. Use the *fromJsonDocument* method to convert *docOut* back to a Java domain object of the specified *type*, and *return* the result from the method.

```
public <T extends Entity> T update(T entity, Class<? extends T> type) {
        JsonDocument docIn = toJsonDocument(entity);
        JsonDocument docOut = bucket.replace(docIn);
        return fromJsonDocument(docOut, type);
}
```

7. (Optional) Review the *fromJsonDocument* method.

B. Consider Couchbase write behavior

8. Run the *lab-07* project. You should see this console output:

9. In the Couchbase administration console, open the *customer360* data bucket, and review the documents and IDs it contains.

10. Select Edit Document for *customer::rred33*, and copy its value.

11. In Firefox, launch HTTP Tool, and configure this PUT request to update the existing document.

  ❏ HTTP Verb (Method): PUT
  ❏ URL for SparkJava: *http://localhost:4567*
  ❏ Base application URL: */customer360*
  ❏ Controller extension: */customer*
  ❏ Document ID: *customer::rred33*

  `http://localhost:4567/customer360/customer/`**`customer::rred33`**

12. Paste the copied document in the Request Body Content field.
13. In the Body Content field, modify "line1" to a different value, and add a "line2" value.

Request

```
http://localhost:4567/customer360/customer/customer::rred33

PUT    ⬍   Submit   Reset

  Headers      Body content

{
  "firstName": "Ricky",
  "lastName": "Red",
  "cas": 0,
  "created": "2015-06-24T15:27:04-0700",
  "id": "customer::rred33",
  "billingAddress": {
      "state": "OR",
      "city": "Portland",
      "line1": "2345 Six St.",
      "line2": "Apartment 1",
      "postalCode": "97203"
  },
  "userName": "rred33",
  "type": "com.couchbase.customer360.domain.Customer",
  "updated": "2015-06-24T15:27:04-0700",
  "email": "rickyred33@mailinator.com"
}
```

14. You should see the modified document returned in the Response Body Content.

Response

```
  Headers     Body content     Status (200)

Copy to clipboard

{
    "id": "customer::rred33",
    "created": "2015-07-01T12:52:35-0700",
    "updated": "2015-07-01T12:52:35-0700",
    "cas": 0,
    "email": "rickyred33@mailinator.com",
    "userName": "rred33",
    "firstName": "Ricky",
    "lastName": "Red",
    "billingAddress": {
        "line1": "2345 Six St.",
        "line2": "Apartment 1",
        "city": "Portland",
        "state": "OR",
        "postalCode": "97203"
    },
    "type": "com.couchbase.customer360.domain.Customer"
}
```

15. Open a new browser tab, and GET the modified document using the same URL. You should see the replaced ("updated") values.

http://localhost:4567/customer360/customer/**customer::rred33**


```
{
    "id" : "customer::rred33",
    "created" : "2015-06-24T15:27:04-0700",
    "updated" : "2015-06-24T15:27:04-0700",
    "cas" : 0,
    "email" : "rickyred33@mailinator.com",
    "userName" : "rred33",
    "firstName" : "Ricky",
    "lastName" : "Red",
    "billingAddress" : {
        "line1" : "2345 Six St.",
        "line2" : "Apartment 1",
        "city" : "Portland",
        "state" : "OR",
        "postalCode" : "97203"
    },
    "type" : "com.couchbase.customer360.domain.Customer"
}
```

16. In HTTP Tool, configure and submit a PUT request to update a <u>non-existing</u> document, using the following settings. However, <u>you should see no response in HTTP Tool</u>.

- ❏ HTTP Verb (Method): PUT
- ❏ URL for SparkJava: *http://localhost:4567*
- ❏ Base application URL: */customer360*
- ❏ Controller extension: */customer*
- ❏ Document ID: ***customer::kkohl44***

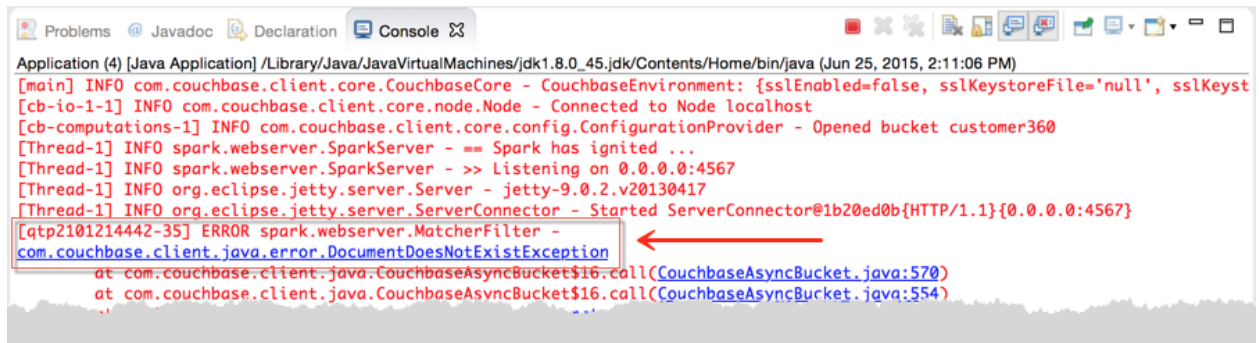http://localhost:4567/customer360/customer/**customer::kkohl44**

Body Content (note, you can copy and paste this code from this document):

```
{
    "id" : "customer::kkohl44",
    "firstName" : "Krishna",
    "lastName" : "Kohl"
}
```

Note, due to the way this particular application handles incoming URLs and JSON, you <u>must</u> add a document here, with (at least) an ID value. The ID value is written into the *JsonDocument* passed to the *replace* operation.

17. In Eclipse, look at the Console. You should see a *DocumentDoesNotExistException*.
    You cannot update a non-existent document. While this is expected, it adds complexity.
    The client must know whether a record exists to determine whether to insert or update it.

Note, while the code in *lab-06* is not configured to handle exceptions, it is easy to do so.
Examine and run the corresponding code in *final* project, which <u>does</u> handle and report
exceptions, including this exception, through the REST API.

18. Stop the running application.

End of lab

Lab 8 - Upsert (insert or replace) a document

Objectives

A	Insert or replace a document in a single operation
B	Modify the customer360 application API to accommodate upsert logic
C	Observe the behavior of multiple PUT operations for the same document

In this lab, you modify the current application logic - which either inserts ("creates") or replaces ("updates") a document - with a single operation that handles both. This is possible, because of the append-only nature of Couchbase write operations.

A. Insert or replace ("update") a document in a single upsert operation

1. In Eclipse, open the *lab-08* project, and close unrelated projects.
2. In *CouchbaseRepository.java*, locate the *upsert* method, review its signature, and delete the *TODO* and placeholder exception. Notice the method receives a Java domain object, named *entity*, along with a reference to its *type*.

```
public <T extends Entity> T upsert(T entity, Class<? extends T> type) {

}
```

3. Use the *toJsonDocument* method to convert the entity passed to this method into a *JsonDocument*, named *docIn*.

```
public <T extends Entity> T upsert(T entity, Class<? extends T> type) {
    JsonDocument docIn = toJsonDocument(entity);

}
```

4. (Optional) Review the *toJsonDocument* method.
5. Pass *docIn* to the *upsert* method of the *bucket*, and assign the result to a *JsonDocument* named *docOut*.

```
public <T extends Entity> T upsert(T entity, Class<? extends T> type) {
    JsonDocument docIn = toJsonDocument(entity);
    JsonDocument docOut = bucket.upsert(docIn);

}
```

6. Use the *fromJsonDocument* method to convert *docOut* back to a Java domain object of the specified *type*, and *return* the result from the method.

```java
public <T extends Entity> T upsert(T entity, Class<? extends T> type) {
        JsonDocument docIn = toJsonDocument(entity);
        JsonDocument docOut = bucket.upsert(docIn);
        return fromJsonDocument(docOut, type);
}
```

7. (Optional) Review the *fromJsonDocument* method.

B. Modify the customer360 application API to accommodate upsert logic

8. In *BaseController.java*, modify the logic handling POST requests to call *upsert* method of the *CouchbaseRepository* instance named *repo*, rather than its *create* method.

```java
if(!methodList.contains(HttpMethod.post)) {
        Spark.post(baseURL, new Route() {
                @Override
                public Object handle(Request request, Response response) {
                        T entity = converter.fromJson(request.body(), type);
                        contentLocation(response, baseURL + "/" + entity.getId());
                        // T result = repo.create(entity, type);
                        T result = repo.upsert(entity, type);
                        return ok(response, result);
                }
        });
}
```

9. Modify the logic handling PUT requests to call the *upsert* method of the *CouchbaseRepository* instance named *repo*, rather than its *create* method.
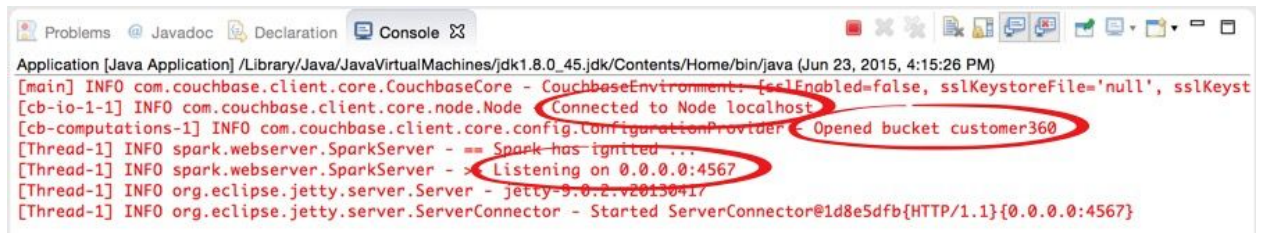
```java
if(!methodList.contains(HttpMethod.put)) {
        Spark.put(baseURL + "/:id", new Route() {
                @Override
                public Object handle(Request request, Response response) {
                        T entity = converter.fromJson(request.body(), type);
                        if(!request.params(":id").equals(entity.getId())) {
                                response.body("Invalid ID specified");
                                return error(response, HttpStatus.BAD_REQUEST_400);
                        }
                        contentLocation(response, baseURL + "/" + entity.getId());
                        // T result = repo.update(entity, type);
                        T result = repo.upsert(entity, type);
                        return ok(response, result);
                }
        });
}
```

C. Observe the behavior of multiple PUT operations for the same document

10. In the Couchbase administration console, verify that the *customer::kkohl44* document was <u>not</u> accidentally added in the prior lab. If so, delete it.
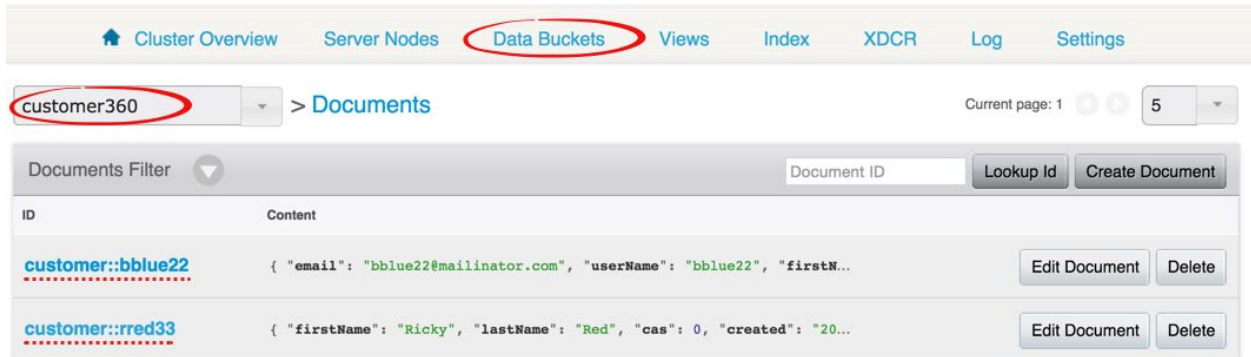
11. Run the *lab-08* project. You should see this console output:

12. In HTTP Tool, configure and submit the same PUT request - to update a non-existing document - which threw an exception in *lab-06*.
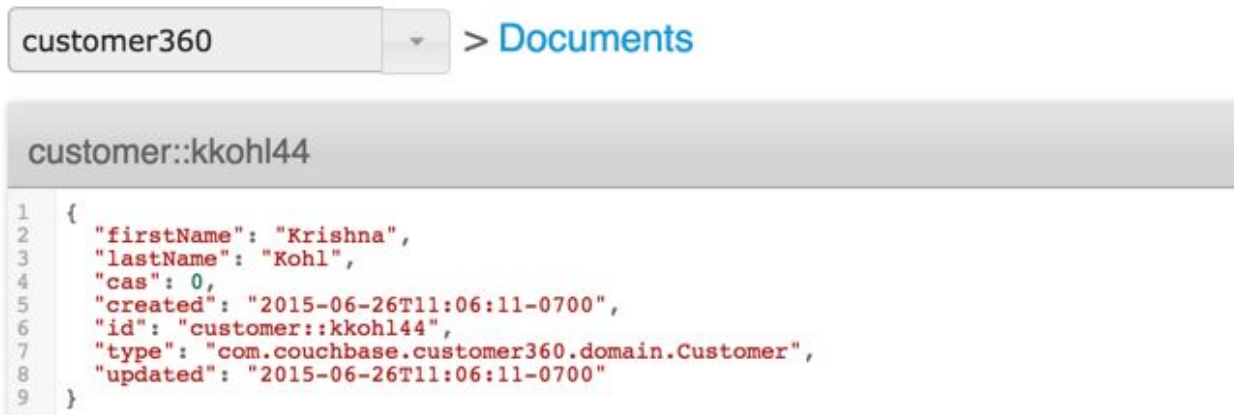
- ❏ HTTP Verb (Method): PUT
- ❏ URL for SparkJava: *http://localhost:4567*
- ❏ Base application URL: */customer360*
- ❏ Controller extension: */customer*
- ❏ Document ID: ***customer::kkohl44***

http://localhost:4567/customer360/customer/**customer::kkohl44**

Body Content (note, you can copy and paste this code from this document):

```
{
  "id" : "customer::kkohl44",
  "firstName" : "Krishna",
  "lastName" : "Kohl"
}
```

13. In the Couchbase administration console, open the *customer360* data bucket, and open the newly added document.
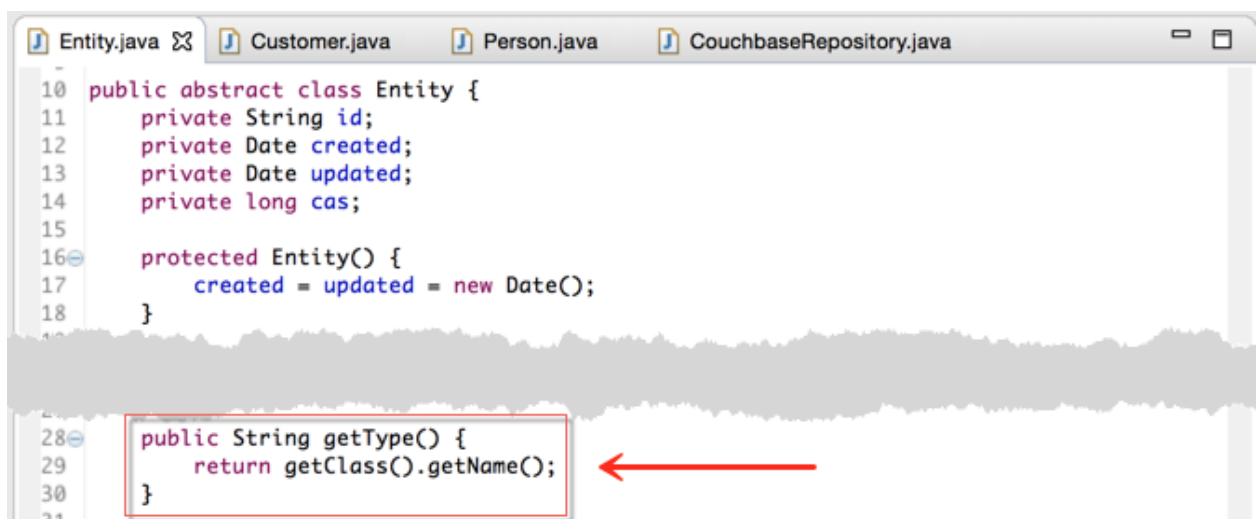
```
customer360          ▾   > Documents

customer::kkohl44

1   {
2     "firstName": "Krishna",
3     "lastName": "Kohl",
4     "cas": 0,
5     "created": "2015-06-26T11:06:11-0700",
6     "id": "customer::kkohl44",
7     "type": "com.couchbase.customer360.domain.Customer",
8     "updated": "2015-06-26T11:06:11-0700"
9   }
```

14. (Optional) Open and briefly review the *Customer.java*, *Person.java*, and *Entity.java* classes. Recall the *toJsonDocument* method in *CouchbaseRepository.java*, and consider how the *type* value became part of the *customer360* documents.

```
J Entity.java ✕   J Customer.java   J Person.java   J CouchbaseRepository.java        ▭ ▢

10  public abstract class Entity {
11      private String id;
12      private Date created;
13      private Date updated;
14      private long cas;
15
16      protected Entity() {
17          created = updated = new Date();
18      }

28      public String getType() {
29          return getClass().getName();       ⬅
30      }
31
```

Note, a document is <u>any</u> binary value up to 20mb. Structure and content are up to you.

15. In HTTP Tool, modify the Body Content of the prior PUT request to add additional fields, then submit the request.

    http://localhost:4567/customer360/customer/**customer::kkohl44**

    Body Content:

    ```
    {
      "id": "customer::kkohl44",
      "email": "krishnakohl44@mailinator.com",
      "userName": "kkohl44",
      "firstName": "Krishna",
      "lastName": "Kohl",
      "billingAddress":
        {
          "line1": "3456 Seven St.",
          "city": "Houston",
          "state": "TX",
          "postalCode": "77386"
        }
    }
    ```

16. In the Couchbase administration console, open the *customer360* data bucket, and open (or refresh the view of) the newly added document. Notice the same PUT operation now creates or updates the document, as appropriate.
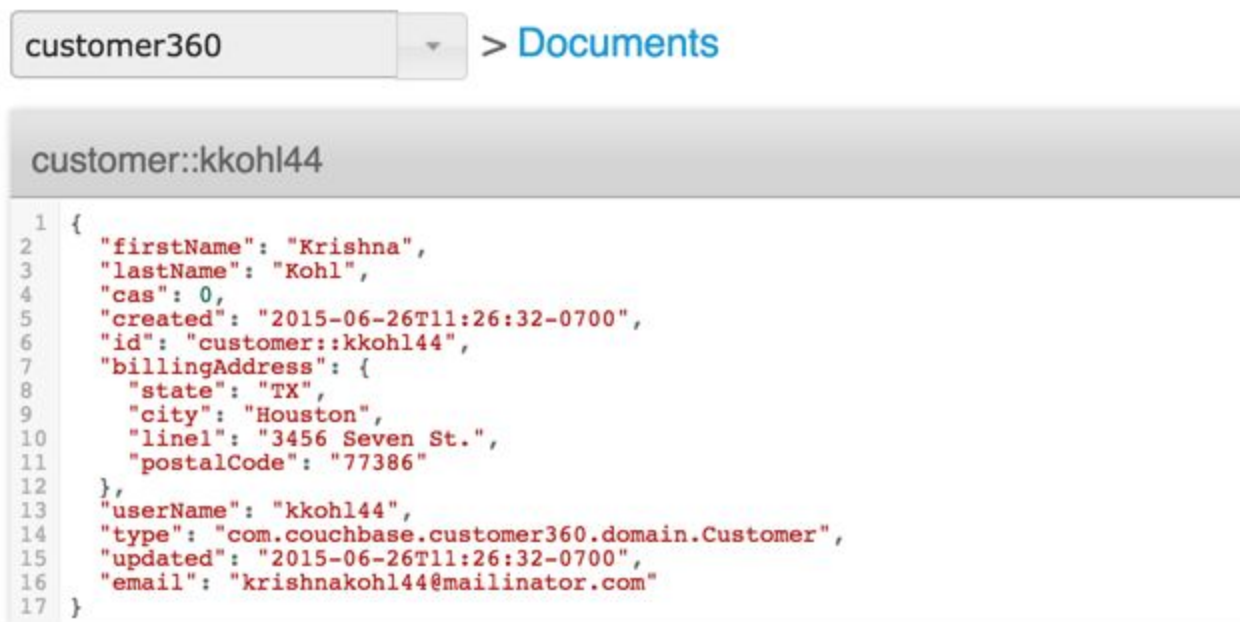
17. Stop the running application.

End of lab

Lab 9 - Remove ("delete") a document

Objectives

A	Remove an existing document
B	Observe the behavior when attempting to remove a non-existing document

In this lab, you implement and test the code to remove an existing document. You also observe the exception thrown when attempting to remove a non-existing document.

Note, in the *lab-08* project, the code has been restored to its state in *lab-06*. You must issue a POST request to insert ("create") a document, and a PUT request to replace ("update") it.

A. Remove an existing document

1. In Eclipse, open the *lab-09* project, and close unrelated projects.
2. In *CouchbaseRepository.java*, locate the *delete* method, review its signature, and delete the *TODO* and placeholder exception. Notice the method receives a Java domain object, named *entity* but, unlike related methods, no reference to its *type*.

```
public <T extends Entity> void delete(T entity) {

}
```

3. Convert the *entity* passed to the *delete* method to a *JsonDocument*.

```
public <T extends Entity> void delete(T entity) {
        JsonDocument doc = toJsonDocument(entity);

}
```

4. Use the *remove* method of the *bucket* reference to "delete" this document.

```
public <T extends Entity> void delete(T entity) {
        JsonDocument doc = toJsonDocument(entity);
        bucket.remove(doc);
}
```

Recall from the video discussion that documents are marked for removal ("tombstoned"), and rendered immediately unavailable. However, they are not physically removed until the next compaction.

5. Run the *lab-09* project. You should see this console output:

6. In HTTP Tool, submit a DELETE request for a document with ID *customer::kkohl44*.

   http://localhost:4567/customer360/customer/**customer::kkohl44**

   Body Content:

   ```
   {
     "id": "customer::kkohl44"
   }
   ```

   Note, the particular REST API implemented in this code requires the ID on the URL, or not, as specified in *BaseController.java*. However, the Couchbase SDK only requires the ID of the *JsonDocument* passed through it to Couchbase Server. Ultimately, you'll decide how to implement your own API.
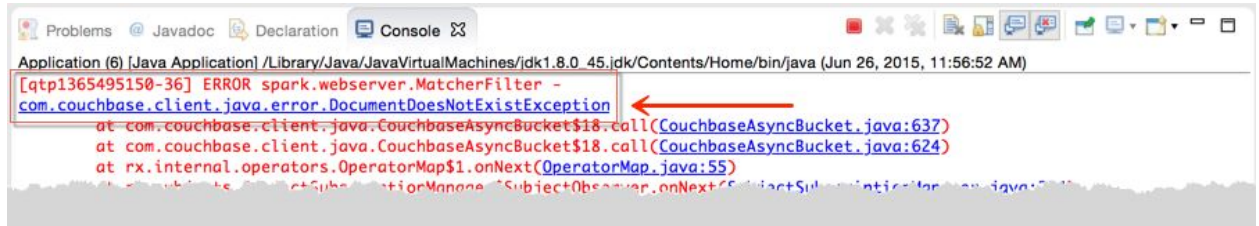
   Note, you could also simply call *bucket.remove(id)*, passing only the document ID itself.

7. In the Couchbase administration console, open the *customer360* data bucket and verify the *customer::kkohl44* document has been removed.

B. Observe the behavior when attempting to remove a non-existing document

8. In HTTP Tool, submit the DELETE request for *customer::kkohl44* a second time.
9. In Eclipse, you should see a *DocumentDoesNotExistException*.

9. Stop the running application.

End of Lab

Lab 10 - Implement and use a cluster-wide counter

Objectives

A	Implement a counter
B	Use a counter to create unique keys
C	Observe auto-incremented ID values

In this lab, you implement a counter, and related code, to generate unique keys following a similar pattern to that used for the manually created keys earlier in this course. The Java domain type is used, followed by a counter value generated atomically by Couchbase server.

A. Implement a named counter

1. In Eclipse, open the *lab-10* project, and close unrelated projects.
2. In *CouchbaseRepository.java*, locate the *getNextId* method, review its signature, read and remove its TODO and placeholder exception. Notice that it receives a *type*, an initial value, and a value by which to increment. Do not delete the assignment of the modified simple name of the *type* to a local String variable called *name*.

   ```
   private <T extends Entity> String getNextId(Class<T> type, long incr,
           long init) {
           String name = type.getSimpleName().toLowerCase();

   }
   ```

3. Use the *counter* method of the *bucket* to create a new counter, concatenating the literal prefix *"counter::"* to the *name* variable..
4. Pass the increment parameter, *incr*, to set how the value will change with each use.
5. Pass the initial value parameter, *init*, to initialize the counter <u>if</u> it does not yet exist.
6. Assign the result to a local *JsonLongDocument* variable named *doc*.

   ```
   private <T extends Entity> String getNextId(Class<T> type, long incr,
           long init) {
           String name = type.getSimpleName().toLowerCase();
           JsonLongDocument doc =
                   bucket.counter("counter::" + name, incr, init);

   }
   ```

7. Extract the counter value from the *content* of *doc*, as a *String*, and combine it with the *name* variable, followed by the literal "::".
8. *Return* this value from the method.

```java
private <T extends Entity> String getNextId(Class<T> type, long incr,
        long init) {
    String name = type.getSimpleName().toLowerCase();
    JsonLongDocument doc =
            bucket.counter("counter::" + name, incr, init);
    return name + "::" + doc.content().toString();
}
```

B. Use a counter to create unique keys

9. In the *create* method - above the existing code converting and inserting the *entity* - review the pre-written condition, using the *isBlank* method, to test whether an *ID* has been assigned to the entity being passed into this method for creation.
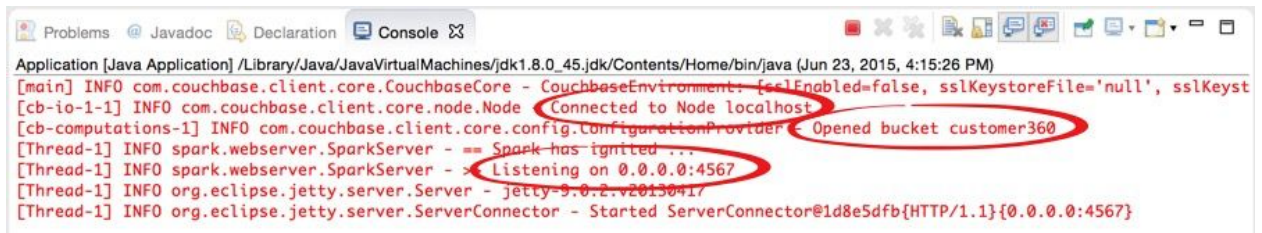
```java
public <T extends Entity> T create(T entity, Class<? extends T> type) {
    if(isBlank(entity.getId())) {

    }
    JsonDocument docIn = toJsonDocument(entity);
    JsonDocument docOut = bucket.insert(docIn);
    return fromJsonDocument(docOut, type);
}
```

10. If the *entity* has not been assigned an *ID*, use the *getNextId* method to get the next atomically unique value for this Java domain type, specifying *1* as the increment value, and *100* as the initial value, if the counter does not yet exist.
11. Set this new value as the *ID* for this entity.

```java
public <T extends Entity> T create(T entity, Class<? extends T> type) {
    if(isBlank(entity.getId())) {
            String id = getNextId(type, 1, 100);
            entity.setId(id);
    }
    JsonDocument docIn = toJsonDocument(entity);
    JsonDocument docOut = bucket.insert(docIn);
    return fromJsonDocument(docOut, type);
}
```

12. Run the *lab-010* project. You should see this console output:

C. Observe auto-incremented ID values

13. In HTTP Tool, POST the following document, which has no ID:

    `http://localhost:4567/customer360/customer`

    Body Content:

    ```
    {
        "email": "yolandayellow55@mailinator.com",
        "userName": "yyellow55",
        "firstName": "Yolanda",
        "lastName": "Yellow",
        "billingAddress": {
            "line1": "QRST Youvee St.",
            "city": "Seattle",
            "state": "WA",
            "postalCode": "98102"
        }
    }
    ```

14. You should see the following copy of the newly inserted document, returned as the Body Content of the response.

15. Modify the Request Body Content, to post a second document (if re-using the prior document, be sure to remove the ID value).

```
{
    "email": "oscarorange66@mailinator.com",
    "userName": "oorange66",
    "firstName": "Oscar",
    "lastName": "Orange",
    "billingAddress": {
        "line1": "678 Ninth Ave.",
        "city": "Seattle",
        "state": "WA",
        "postalCode": "98102"
    }
}
```

16. You should see the new document in the response, with an ID incremented from the previously posted document.

```
{
    "id": "customer::101",
    "created": "2015-06-29T15:52:26-0700",
    "updated": "2015-06-29T15:52:26-0700",
    "cas": 0,
    "email": "oscarorange66@mailinator.com",
    "userName": "oorange66",
    "firstName": "Oscar",
    "lastName": "Orange",
    "billingAddress": {
        "line1": "678 Ninth Ave.",
        "city": "Seattle",
        "state": "WA",
        "postalCode": "98102"
    },
    "type": "com.couchbase.customer360.domain.Customer"
}
```

17. In the Couchbase administration console, open the *customer360* data bucket, and notice the *counter::customer* document, and its value (the actual counter value you see may vary, depending on how many documents you ultimately posted).

Documents Filter ▼		Document ID	Lookup Id	Create Document
ID	**Content**			
counter::customer	102		Edit Document	Delete
customer::100	{ "firstName": "Yolanda", "lastName": "Yellow", "cas": 0, "created"...		Edit Document	Delete

18. Stop the running application.

End of Lab

Lab 11 - Implement optimistic locking using CAS

Objectives

A	Implement optimistic locking by tracking CAS values on client
B	Test the behavior seen when attempting to change a locked document

In this exercise, you cause CAS values to be persisted as Java domain types are converted to and from the JsonDocument type. You then test how locked document

A. Implement optimistic locking by tracking CAS values on client

1. In Eclipse, open the *lab-11* project, and close unrelated projects.
2. In *CouchbaseRepository.java*, locate the *toJsonDocument* method, review its signature and existing code, remove the TODO comment, and comment out the line which creates the *JsonDocument*, named *doc*, from the Java domain object passed in as *source*.
3. Modify the *toJsonDocument* method to pass the CAS value of the Java domain object passed to this method for conversion, to the *JsonDocument.create* factory method, in addition to the *ID* and *content*.

```java
protected <T extends Entity> JsonDocument toJsonDocument(T source) {
  if (source == null) {
    throw new IllegalArgumentException("entity is null");
  }
  String id = source.getId();
  if (id == null) {
    throw new IllegalStateException("entity ID is null");
  }
  try {
    JsonObject content =
      transcoder.stringToJsonObject(converter.toJson(source));
    // JsonDocument doc = JsonDocument.create(id, content);
    JsonDocument doc = JsonDocument.create(id, content, source.getCas());
    return doc;
  } catch (Exception e) {
    throw new RepositoryException(e);
  }
}
```
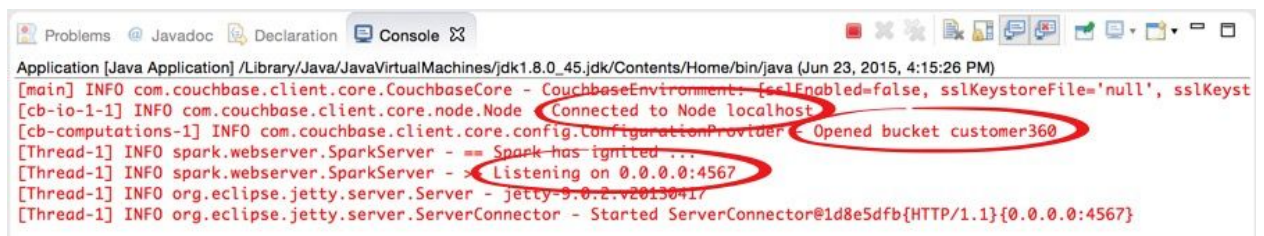
4. Locate the *fromJsonDocument* method, review its signature and existing code, and remove the TODO comment.

5. Use the *setCas* method of the *result* object to assign this object the CAS value of the *JsonDocument* being passed to this method for conversion back to a Java domain type.

```
protected <T extends Entity> T fromJsonDocument(JsonDocument doc,
  Class<T> type) {
      if (doc == null) {
            throw new IllegalArgumentException("document is null");
      }
      JsonObject content = doc.content();
      if (content == null) {
            throw new IllegalStateException("document has no content");
      }
      if (type == null) {
            throw new IllegalArgumentException("type is null");
      }
      T result = converter.fromJson(content.toString(), type);
      result.setCas(doc.cas());
      return result;
}
```

B. Test the behavior seen when attempting to change a locked document

6. Run the *lab-11* project. You should see this console output:

7. In the Couchbase administration console, open the *customer360* data bucket, and note the ID of one of the documents generated in the prior exercise.

8. In HTTP Tool, GET the document using its document ID.

   http://localhost:4567/customer360/customer/**customer::101**

   ```
   {
       "id": "customer::101",
       "created": "2015-06-30T16:01:13-0700",
       "updated": "2015-06-30T16:01:13-0700",
       "cas": 627903408373760,
       "email": "oscarorange66@mailinator.com",
       "userName": "oorange66",
       "firstName": "Oscar",
       "lastName": "Orange",
       "billingAddress": {
           "line1": "678 Ninth Ave.",
           "city": "Seattle",
           "state": "WA",
           "postalCode": "98102"
       },
       "type": "com.couchbase.customer360.domain.Customer"
   }
   ```

   Notice that, due to the code modified above, the CAS value now being persisted in the client code (e.g., *JsonDocument*) is also written into the document. Normally, CAS is present only as metadata within the *JsonDocument* object, and in Couchbase server.

9. Copy the Response Body Content.
10. To simulate a second client, <u>leave the first tab open</u>, and open HTTP Tool in a <u>second</u> browser tab.
11. In HTTP Tool, paste the copied document in the Request Body Content field, modify the *billingAddress* to different values, and PUT the change to update the document.

   Note, **do not change the CAS value**, as it represents the current version of this document. Without it, or with a different value, this replace operation will fail.

   http://localhost:4567/customer360/customer/customer::101

   ```
   {
       "id": "customer::101",
       "created": "2015-06-30T16:01:13-0700",
       "updated": "2015-06-30T16:01:13-0700",
       "cas": [do not change this value],
       "email": "oscarorange66@mailinator.com",
       "userName": "oorange66",
       "firstName": "Oscar",
       "lastName": "Orange",
       "billingAddress": {
           "line1": "ABC Dee St.",
           "city": "Portland",
           "state": "OR",
           "postalCode": "97203"
       },
       "type": "com.couchbase.customer360.domain.Customer"
   }
   ```

12. In the Response Body Content, you should see the changed values, along with a <u>new</u> CAS and updated time stamp.

```
{
    "id": "customer::101",
    "created": "2015-06-30T16:01:13-0700",
    "updated": "2015-06-30T16:01:13-0700",
    "cas": 628614979584000,
    "email": "oscarorange66@mailinator.com",
    "userName": "oorange66",
    "firstName": "Oscar",
    "lastName": "Orange",
    "billingAddress": {
        "line1": "ABC Dee St.",
        "city": "Portland",
        "state": "OR",
        "postalCode": "97203"
    },
    "type": "com.couchbase.customer360.domain.Customer"
}
```

13. Return to the first browser tab.
14. In the <u>first</u> browser tab, copy the Response Body Content (from the original GET) into the Request Body Content, modify the email name to "TEST", and PUT the change.

    Note, because the prior PUT/replace changed the CAS value, this value no longer matches the current state of the document in Couchbase.

Request

http://localhost:4567/customer360/customer/customer::101
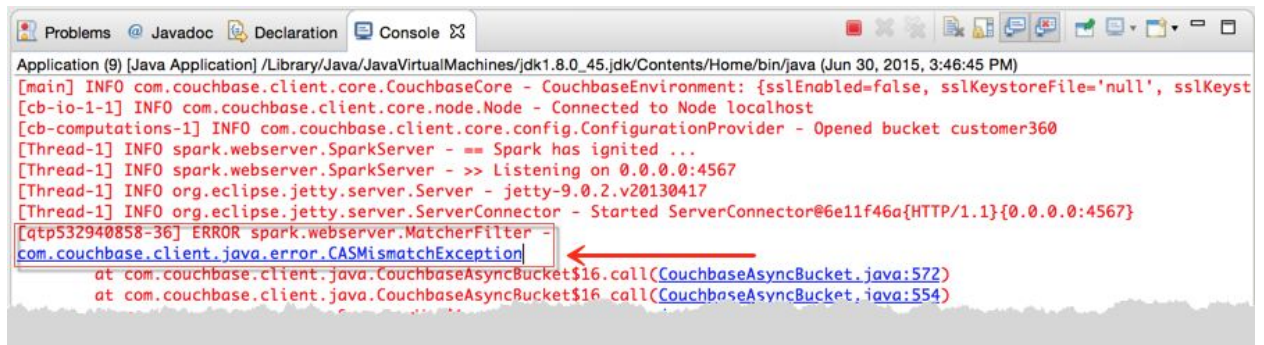
PUT ⬍ Submit Reset

Headers   Body content

```
{
    "id": "customer::101",
    "created": "2015-06-30T16:01:13-0700",
    "updated": "2015-06-30T16:01:13-0700",
    "cas": 627903408373760,
    "email": "TEST@mailinator.com",
    "userName": "oorange66",
    "firstName": "Oscar",
    "lastName": "Orange",
    "billingAddress": {
        "line1": "678 Ninth Ave.",
        "city": "Seattle",
        "state": "WA",
        "postalCode": "98102"
    },
    "type": "com.couchbase.customer360.domain.Customer"
}
```

Note, you should not see any change in the HTTP Tool Response, because an unhandled exception should be thrown by this attempt to PUT a change using an outdated CAS value.

15. In Eclipse, notice the *CASMismatchException* in the Console.

16. (Optional) Open the *lab-final* project, and review the *update* method in *CouchbaseRepository.java*, to see how this exception could be handled and reported back via the REST API.
17. Close the running application.

End of lab