Company Name: FirstEigen

Name: Aishwarya Patil

Joining date: 27th December, 2021

College: AISSMS COE, Pune

Year & Department: Third Year, Computer Dept.

**Data source connection using Couchbase and JAVA**

**Step 1: Install and Start Using the Java SDK with Couchbase Server**

* Download the latest Java LTS version and Couchbase Data Platform.
* Download a zip file with all the dependencies bundled if we wish to manually include the jar files in your classpath.
* Or we can also manually add the dependencies in the pom.xml file with below code:

*<dependencies>*

*<dependency>*

*<groupId>com.couchbase.client</groupId>*

*<artifactId>java-client</artifactId>*

*<version>2.7.23</version>*

*</dependency>*

*</dependencies>*

* Once you have the Java client installed, open your IDE, and try out the following: (Connecting to the Cluster)

*Cluster cluster = Cluster.connect(connectionString, username, password);*

* If you are not using an IDE or are new to Java, the following imports are necessary to build the following snippets:

*import com.couchbase.client.java.\*;*

*import com.couchbase.client.java.kv.\*;*

*import com.couchbase.client.java.json.\*;*

*import com.couchbase.client.java.query.\*;*

**Step 2: Managing Connections using the Java SDK with Couchbase Server**

1. **Connecting to a Cluster:**

A connection to a Couchbase Server cluster is represented by a Cluster object. A Cluster provides access to Buckets, Scopes, and Collections, as well as various Couchbase services and management interfaces. The simplest way to create a Cluster object is to call Cluster.connect() with a connection string, username, and password:

*//this tunes the SDK (to customize connection timeout)*

*CouchbaseEnvironment env = DefaultCouchbaseEnvironment.builder()*

*.connectTimeout(50000) //10000ms = 10s, default is 5s*

*.build();*

*System.out.println("Create connection");*

*//use the env during cluster creation to apply*

*Cluster cluster = CouchbaseCluster.create(env, "127.0.0.1:8091");*

1. **Opening the Bucket:**

Once a Cluster reference is available, you can open Bucket instances and then perform operations on them:

*System.out.println("Try to openBucket");*

*Bucket bucket1 = cluster.openBucket("example");*

*//you can also force a greater timeout here (cluster.openBucket("beer-sample", 10, TimeUnit.SECONDS))*

1. **Performing the Document operations:**

* **Inserting** a document works like this:

*JsonDocument doc = JsonDocument.create("document\_id", JsonObject.create().put("some", "value"));*

*System.out.println(bucket.insert(doc));*

* If the same code is called again, a DocumentAlreadyExistsException will be thrown. If you don’t care that the document is overridden, you can use **upsert** instead:

*JsonDocument doc = JsonDocument.create("document\_id", JsonObject.empty().put("some", "other value"));*

*System.out.println(bucket.upsert(doc));*

* Finally, a full document can be **replaced** if it existed before. If it didn’t exist, then a DocumentDoesNotExistException will be thrown:

*JsonDocument doc = JsonDocument.create("document\_id", JsonObject.empty().put("more", "content"));*

*System.out.println(bucket.replace(doc));*

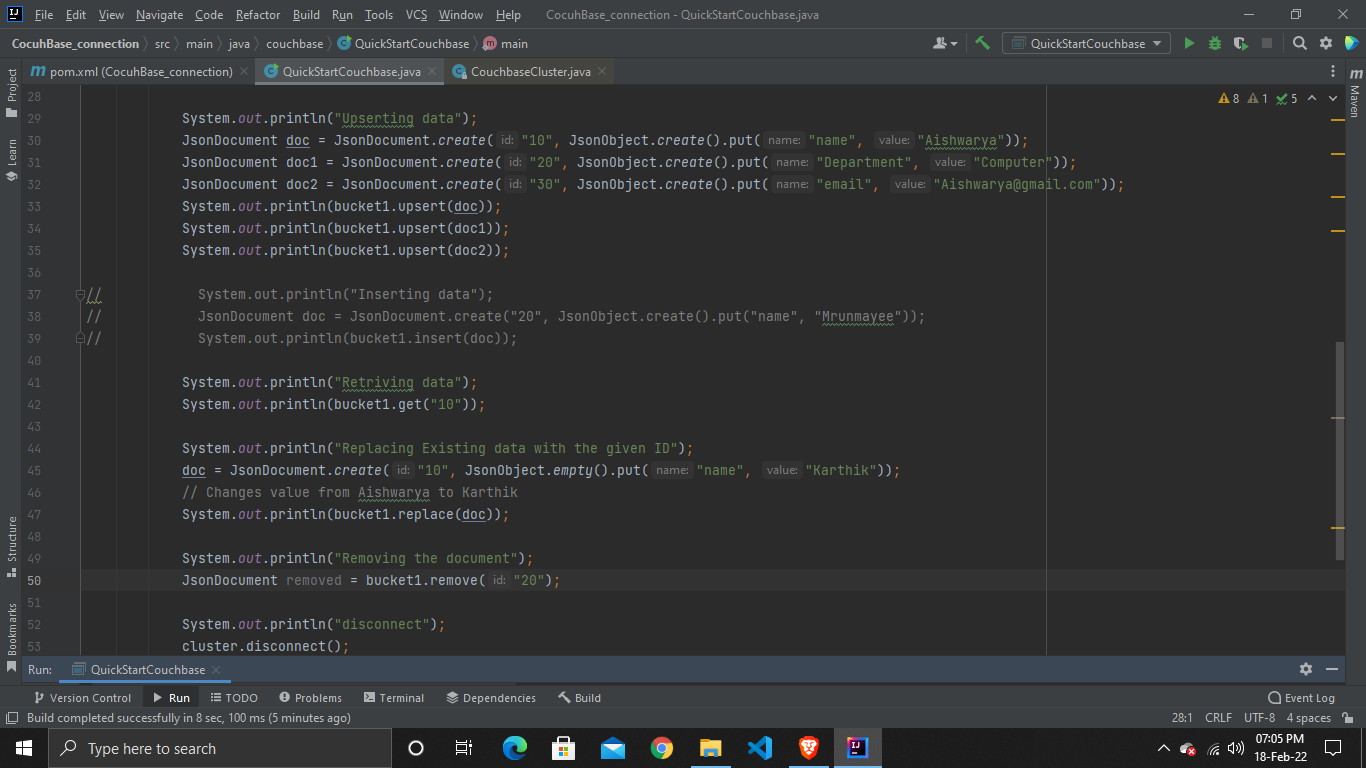
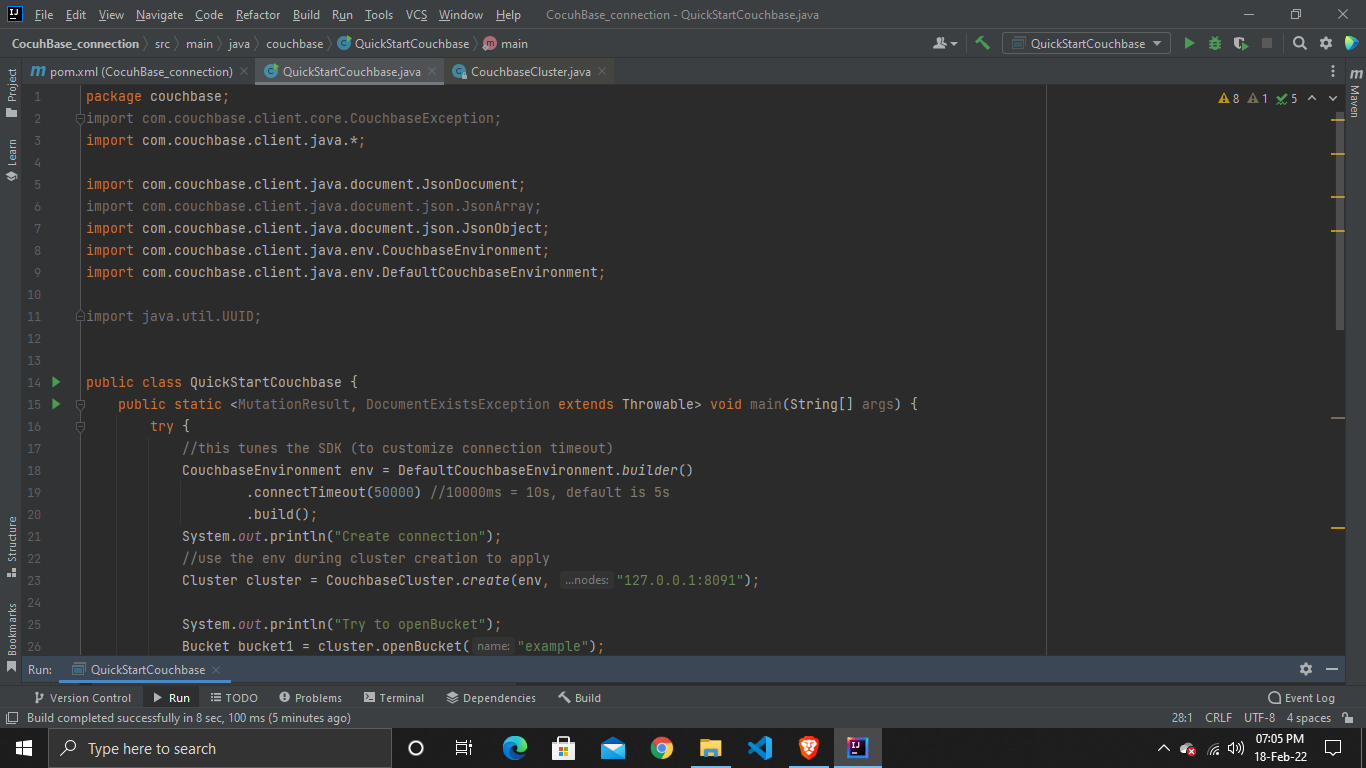
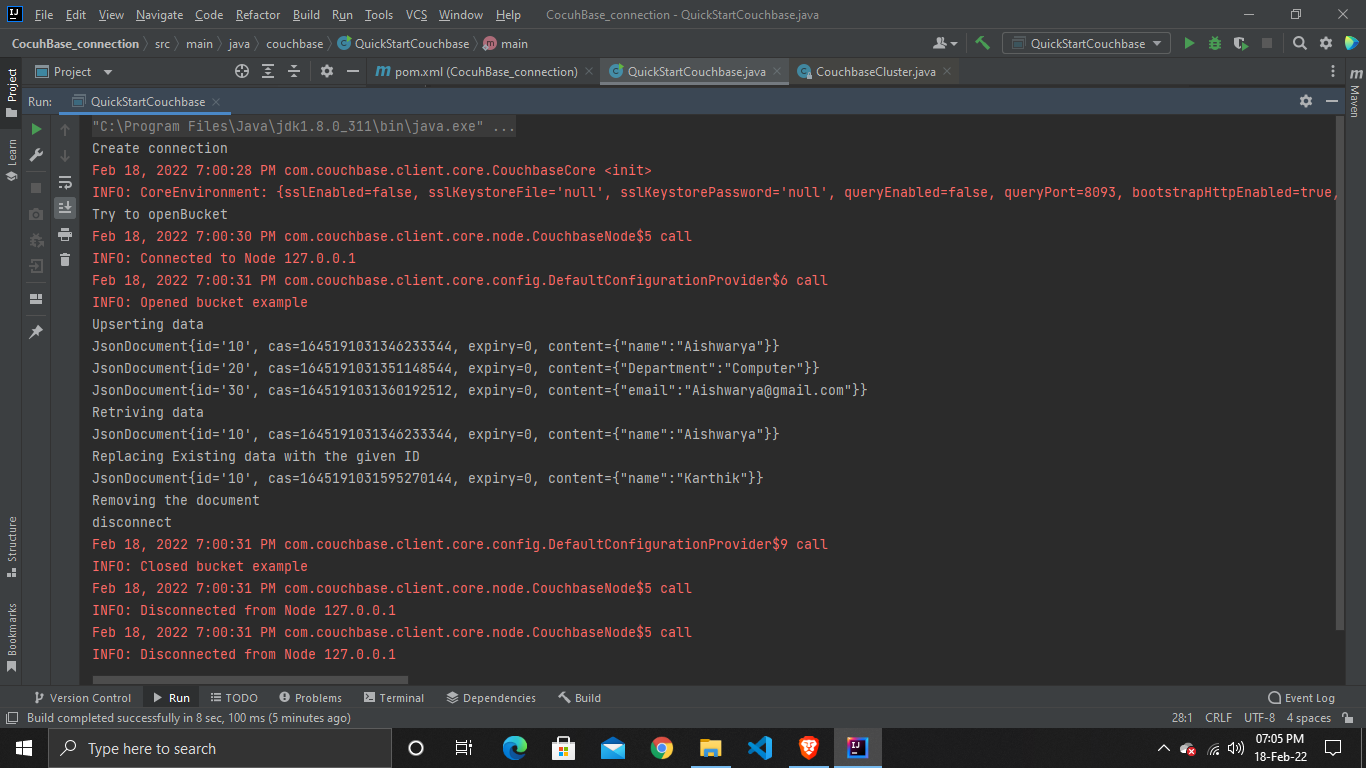
* You can **retrieve** documents using the Bucket.get():

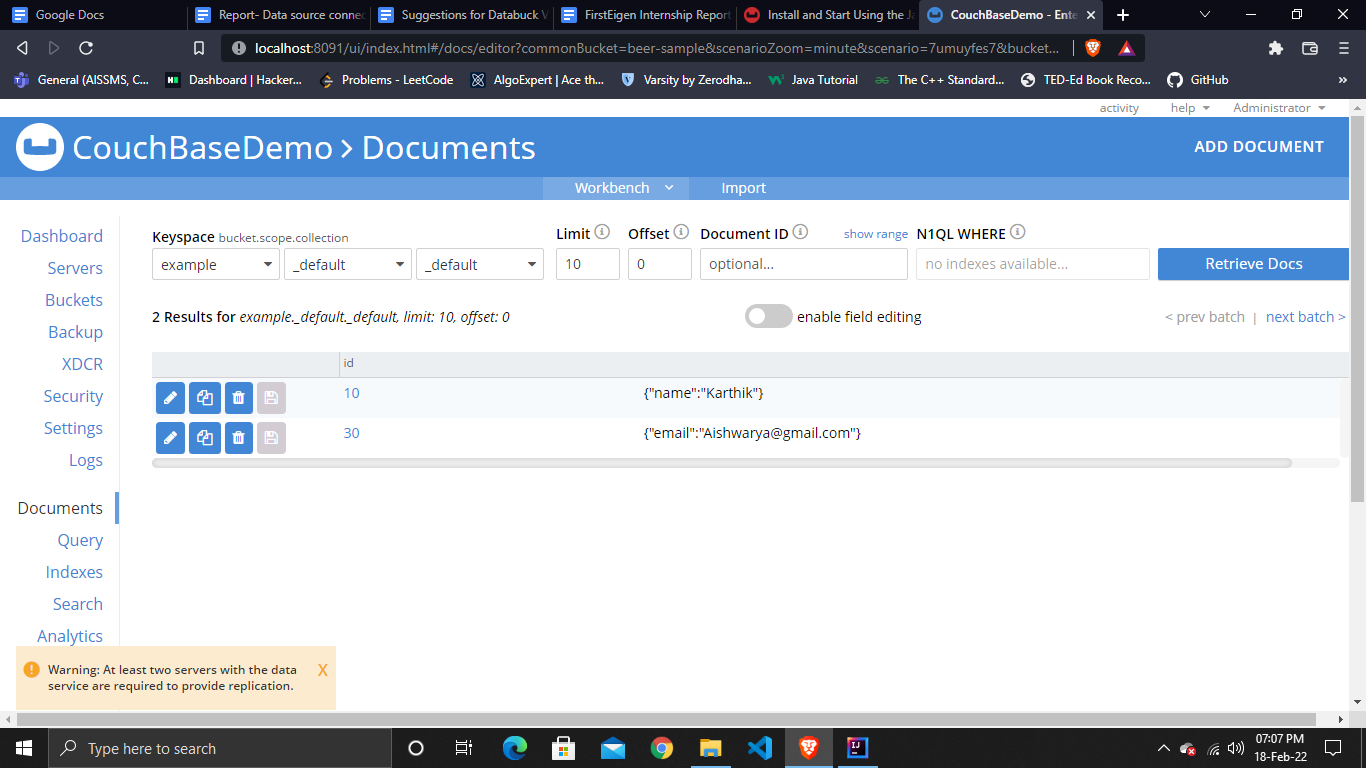
*System.out.println(bucket.get("document\_id"));*

* You can **remove** documents using the Bucket.remove() method. This method takes a single mandatory argument:

*// Remove the document*

*JsonDocument removed = bucket.remove("document\_id");*



**