

Introduction

- ≡ ○ Name Mongo came from word 'Humongous' because it can store lots and lots of data and we can work with it in a very high performant way.
- ≡ ○ Its a popular NoSQL/Non Relational/Key-Value cross platform document oriented database. It provides high performance, high availability and easy scaling.
- ≡ ○ A document oriented database provides APIs or a query language that exposes the ability to query based on the internal structure in the document.
- ≡ ○ Mongo DB documents are composed of field value pairs and have the following structure:
 { field1:value1, field2:value2 }
- ≡ ○ Mongo DB is not a RDBMS system and it does not have any

concept of joins.

Database, Collections & Documents

- ≡ ○ **Database:** Database is a physical container for single or multiple collections. A database can have any number of collections.
Database is a namespace for collections.
- ≡ ○ Each database gets its own set of files on the file system. A Mongo DB server can host multiple such databases inside it.
- ≡ ○ **Collections:** Collection is a group of Mongo DB Documents. You can relate a collection as a Table. Unlike tables, collections do not have any schema definition.
- ≡ ○ Unlike RDBMS database - the collections do not have any concept of JOINS. However we can achieve joins functionality using Aggregations in Mongo.

- ≡ ○ **Documents:** Document in Mongo is a simple JSON key-value pair data. Every document in Mongo has a primary key '_id' with a unique value, which will be added automatically by Mongo DB. Documents have flexible and dynamic schema which are user defined and are not fixed or static.
- ≡ ○ Documents within a collection can have different schema or fields and are related data belonging to a particular subject. Documents can hold any data as long as they are valid data types in Mongo.
- ≡ ○ Each key value pairs are called field in Mongo DB.

Data model design

- ≡ ○ Mongo provides 2 types of data models. Embedded & normalized

data model.

- ≡ ○ In the embedded model you can have all the related data in a single document. For example we can embed employees personal details, contact and address into a single document.
- ≡ ○ In case of normalized data model, we can have separate documents for each of the above details and all the sub document will have a reference to the parent documents _id field value.

Create and Drop Databases

- ≡ ○ To create a database: 'use <database-name>'. The newly created database will not visible unless we insert any document inside it.
- ≡ ○ To remove a database: do.dropDatabase()
- ≡ ○ Also, we can create or drop

databases from Mongo DB Compass tool as well. Also, we can use MongoDB extension in VS code and connect to server.

- ≡ ○ Other list of commands: show databases; db - to show currently connected db;

Create and Drop Collections

- ≡ ○ To create a collection:
`'db.createCollection(name, options)'`
- ≡ ○ To drop a collection:
`'db.collection.drop()'`

Data types in Mongo DB

• Data Types in MongoDB

- BSON
- JSON
- Integer
- Boolean
- Double
- Arrays
- Object
- Null
- Date
- Timestamp
- Object Id
- Code

[Subscribe and Ask your doubts](#)

mongoDB BSON and JSON

• Difference between BSON and JSON

ARC Tutorials

- JSON based databases usually return query results which can be effortlessly parsed, having modest or nix transformation, straightforwardly by the use of JavaScript along with most well-liked programming languages.
- In the case of MongoDB, data representation is done in JSON document format, but here the JSON is binary-encoded, which is termed as BSON.
- BSON is the extended version of the JSON model, which is providing additional data types, makes performance to be competent to encode and decode in diverse languages and ordered fields.

[Subscribe and Ask your doubts in comments section](#)

Insert/update/delete documents in collections

mongoDB Inserting Documents into collections

- Insert is used for creating new documents inside the collection

ARC Tutorials

- To Insert any document into Collection

- `Db.<collection-name>.insert({ "name": "Arc Tutorials"})`

- To insert many documents at once into collection

- `Db.<collection-name>.insertMany(`
 - `{ "name": "Arc Tutorials"},`
 - `{ "name": "Sai Ram"}``)`

Subscribe and Ask your doubts in comments section

mongoDB Updating Documents into collections

- To Update any document into Collection

ARC Tutorials

- Update can be applied with
 - `update`
 - `updateOne`
 - `updateMany`

- `Db.<collection-name>.update(`
`{ "name": "Arc Tutorials"},`
`{`
 `$set: {`
 `"key": "value"`
 `}`
`)`

Subscribe and Ask your doubts in comments section

mongoDB Delete documents from Collections

- Multiple ways to Delete data from collection

- **deleteOne()** – finds all documents in collection
 - db.collection.find()
 - Example: db.orders.deleteOne({ "_id" : ObjectId("563237a41a4d68582c2509da") });
- **deleteMany({})**
 - Will delete many documents at once
 - When passed with empty curly brace – it will delete all documents in collections
 - Example: db.orders.deleteMany({});

ARC Tutorials

[Subscribe](#) and Ask your doubts in comments section

Projections in Mongo DB

- ≡ ○ In Mongo DB it is nothing but another name for querying fields or showing or hiding. Its like select statement in SQL.
- ≡ ○ We need to specify the field as "1" if we need this in the query result, else specify it as "0".
- ≡ ○ `db.<db_name>.find({"Tax":1, "_id":0});`

mongoDB Conditions in Find Method

- We can use multiple operations inside the Find method
- Using operations we add more search power to Find method
- **Various conditions that can be used are:**
 - Equality
 - Less Than
 - Less than equal
 - Greater Than
 - Greater Than Equal
 - Not Equal

ABC Tutorials

[Subscribe and Ask your doubts in comments section](#)

Aggregations in Mongo DB

≡ ○ The way to use join in Mongo

mongoDB Aggregation in MongoDB

- **What is Aggregation in MongoDB?**
 - Aggregate is very similar to the find command, where you can provide the criteria for your query in the form of JSON documents
 - The key element in aggregation is called the pipeline
 - It also helps us in performing few operations like min, max, sum etc
- **The command to use Aggregation is :**
 - `db.leads.aggregate(pipeline, options)`
- **What's pipeline?**
 - A sequence of data aggregation operations or stages
 - Pipeline is an Array
- **What are options?**
 - Documents can be passed as well

[Subscribe and Ask your doubts in comments section](#)

- **What are valid Aggregate Stages?**

- \$count
- \$group
- \$limit
- \$lookup
- \$match
- \$merge
- \$sort
- \$project
- \$unwind
- \$unset

mongoDB Aggregation Explained in Detail

- **Pipeline definition**

```
pipeline = [  
    { ... },  
    { ... },  
    { ... },  
];
```

ARC Tutorials

Example:

```
var pipeline = [  
    { $group: { "_id": "$city" } }  
    { $sort: { "leadName": 1 } },  
    { $limit: 4 }  
];  
  
db.leads.aggregate(pipeline);
```

[Subscribe and Ask your doubts in comments section](#)

Limit and Skip in Mongo DB

mongoDB Limit and Skip in MongoDB

- We may not always want all documents all the time
 - `db.collection.find().limit(4);`
- We may want to skip some documents we don't need
 - `db.collection.find().skip(3);`
 - Always remember – skip will skip sequentially not random or advanced

Sorting in Mongo DB

mongoDB Sorting in MongoDB

- We will need to sort the record set before passing it to next logical operation
 - To sort we can use the below command
 - `db.collection.find().sort({"leadName": 1 })`
 - 1 : means ascending
 - -1 : means descending

Replication in Mongo DB

≡ ○ Replication is the process of synchronizing data across

multiple servers. It increases data availability with multiple copies of data on different database server

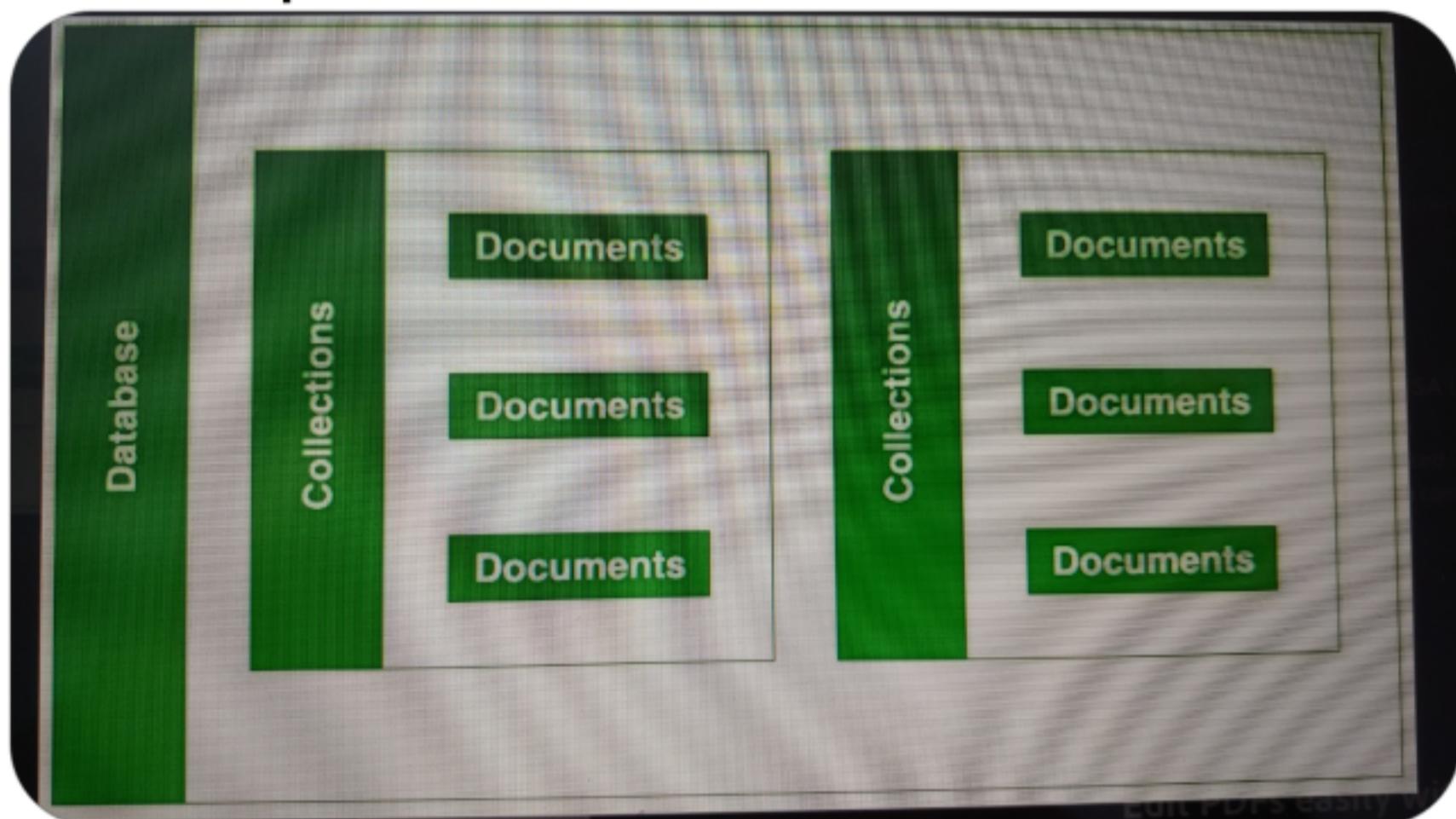
- ≡ ○ **Replica Set:** Mongo DB achieves replication by the use of Replica Set. Replica set is a group of mongod instances that host the same data set. Its a group of 2 or more nodes.
- ≡ ○ In replica set, one node will be the primary node that receives all write operations, and remaining nodes are secondary.
- ≡ ○ All data replicates from primary to secondary. At the time of automatic failover or maintenance, election establishes for primary and a new primary node is selected.
- ≡ ○ We can create such replica set by running the Mongo db with --repSet option.

Sharding in Mongo DB

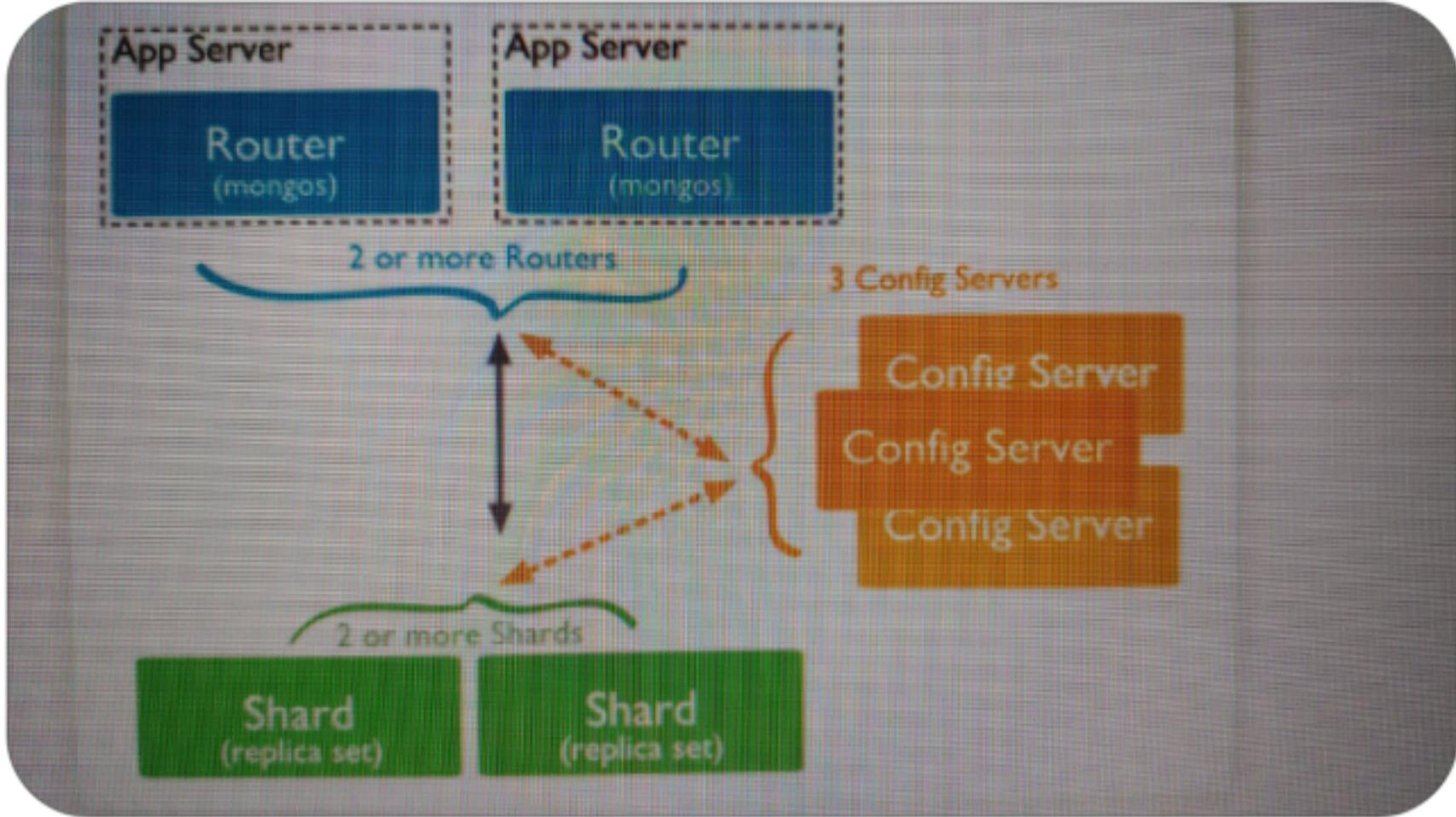
- ≡ ○ Sharding is the process of splitting large data sets into small data sets and storing them across multiple Mongo DB instances and it is Mongo DB's approach to meeting the demands of data growth. As the size of the data increases, a single machine may not be sufficient to store the data nor provide an acceptable read & write throughout.
- ≡ ○ Sometimes the data in mongo DB will be so huge, that queries against such bigbdata can cause a lot of CPU utilization on the server. To tackle this we use Sharding.
- ≡ ○ The collection which could be large in size is actually split across multiple collections or Shards as they are called.

Logically all Shards work as one collection.

- ≡ ○ Sharding solves the problem with horizontal scaling. With sharding, you add more machines to support data growth and the demands of read and write operations.



Shard Cluster



- ≡ ○ Shard: It is the most basic unit of a Shared Cluster that is used to store a subset of the large dataset that has to be devided
- ≡ ○ Config Servers: It stores the metadata of the MongoDB sharded cluster. This metadata consists of information about what subset of data is stored in which Shard. This information can be used to direct user queries accordingly. Each sharded cluster is supposed to have exactly 3 config servers.
- ≡ ○ Query Routers: It can be seen as

Mongo instances that form an interface to the client applications. It is responsible for forwarding user queries to the right Shard.

Backup and Restore

- ≡ ○ To create backup of database in Mongo use the command 'mongodump'.
 - ≡ ○ To restore it we can use the command 'mongorestore'
-

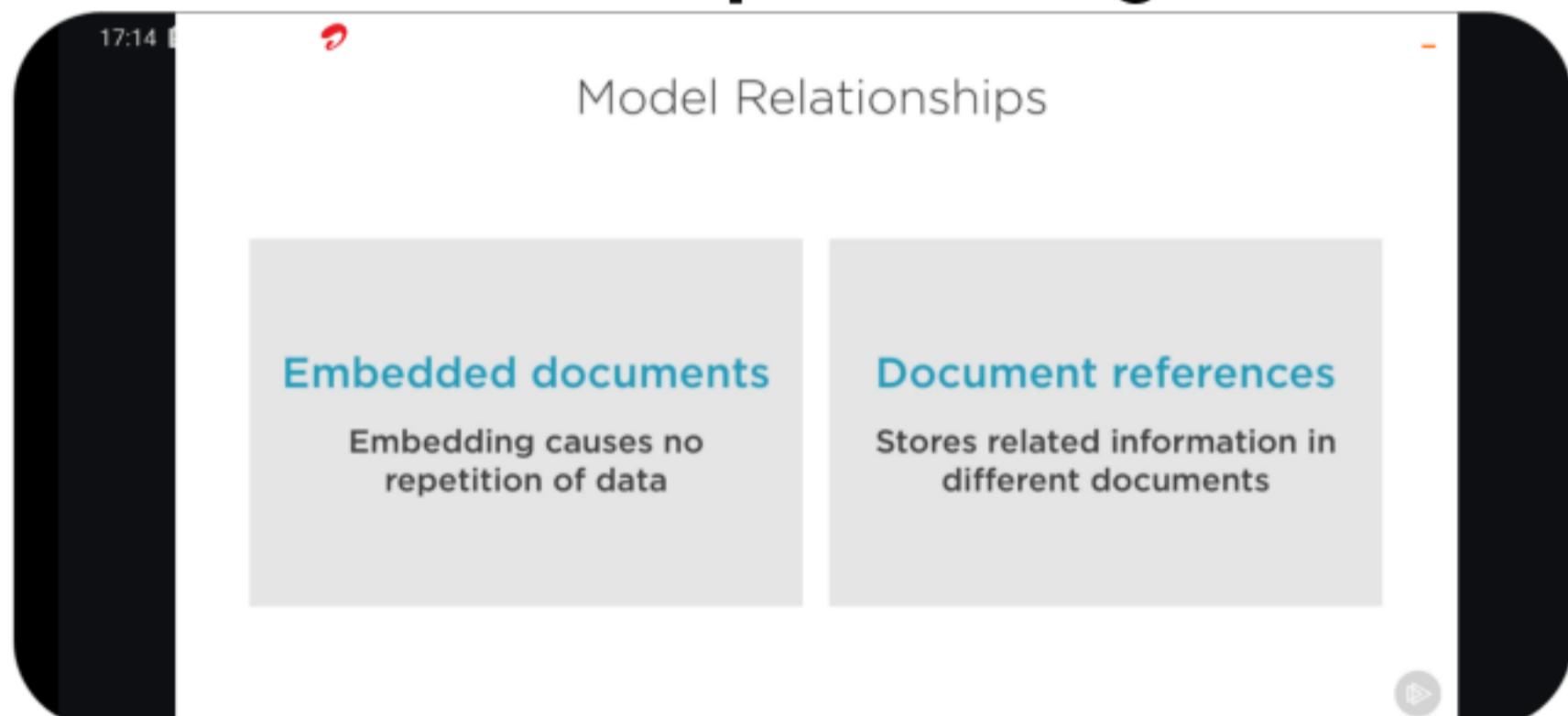
Spring Data MongoDB

- ≡ ○ We can use spring data Mongo plugin to connect our spring boot application to mongo DB.
- ≡ ○ Write java classes with @Document annotation. This will convert java classes to document

in mongo DB.

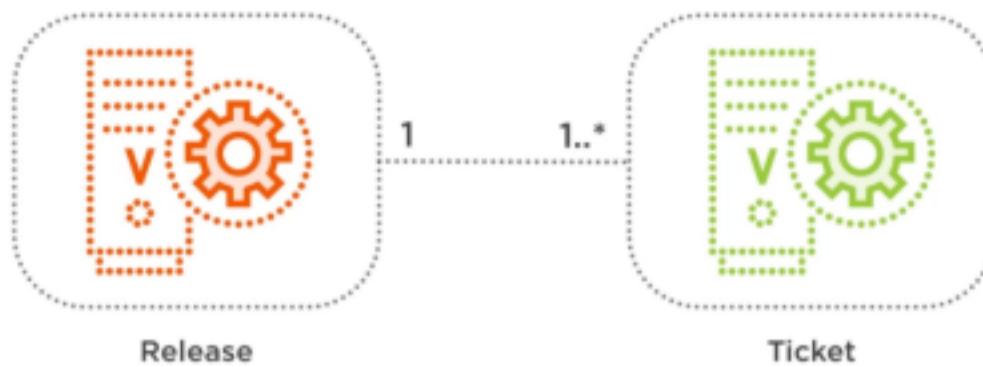
- ≡ ○ Write java class with @Repository annotation which implements MongoRepository, similar way how we use CrudRepository in case of Spring data JPA.
- ≡ ○ Finally, we can use these classes to implement our business logic in service layers.

Model Relationships in Mongo DB



Embedded documents example
One to Many Relationship

Document Relationship



Release.java

```
@Document  
public class Release {  
    @Id  
    private String id;  
    private String name;  
    private String description;  
    private List<Ticket> tickets;  
    ...  
}
```



Document references example
One to one relationship

17:56

```
package com.keysoft.mongodb.model;

import ...

@Document
public class Ticket {

    @Id
    private String id;
    private String title;
    private String description;
    private String appId; ←
    private String status;

    public Ticket() {
    }

    public Ticket(String title, String description, String application_id, String status) {
        this.title = title;
        this.description = description;
        this.appId = application_id;
        this.status = status;
    }
}
```

Mongo Repository

- By using Mongo Repository provided by spring data mongo plugin, we can easily query mongo DB. It provides many build in Mongo DB access methods and also we can write our own custom queries

MongoRepository



```

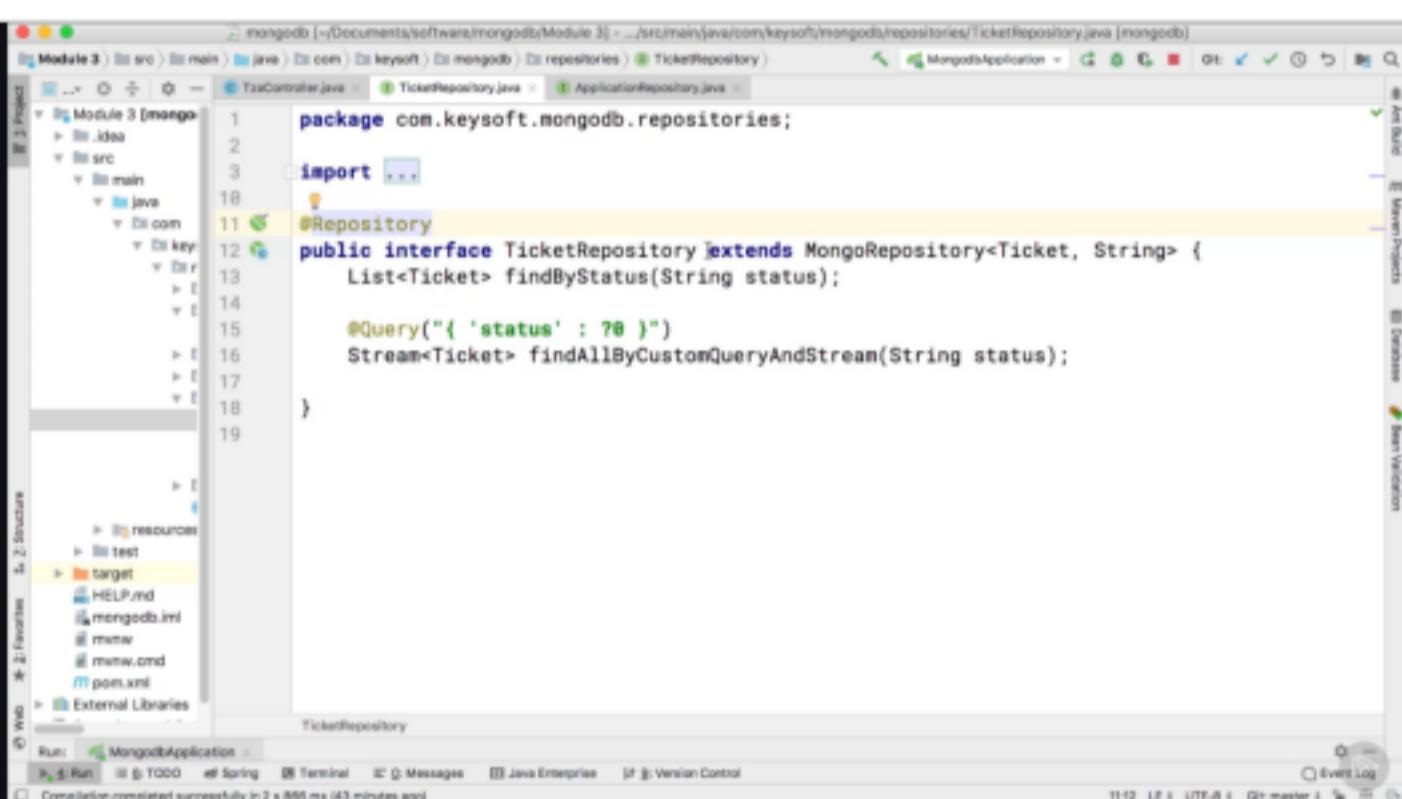
@Repository
public interface TicketRepository
extends MongoRepository<Ticket,
String>
{
    List<Ticket> findByStatus(String
status);

    List<Ticket> findByTitle(String
title);

    List<Ticket> findByAppId(String
appId);
}

```

- Query against MongoDB database
- Define method in repository interface
- Query builder mechanism



Mongo Template

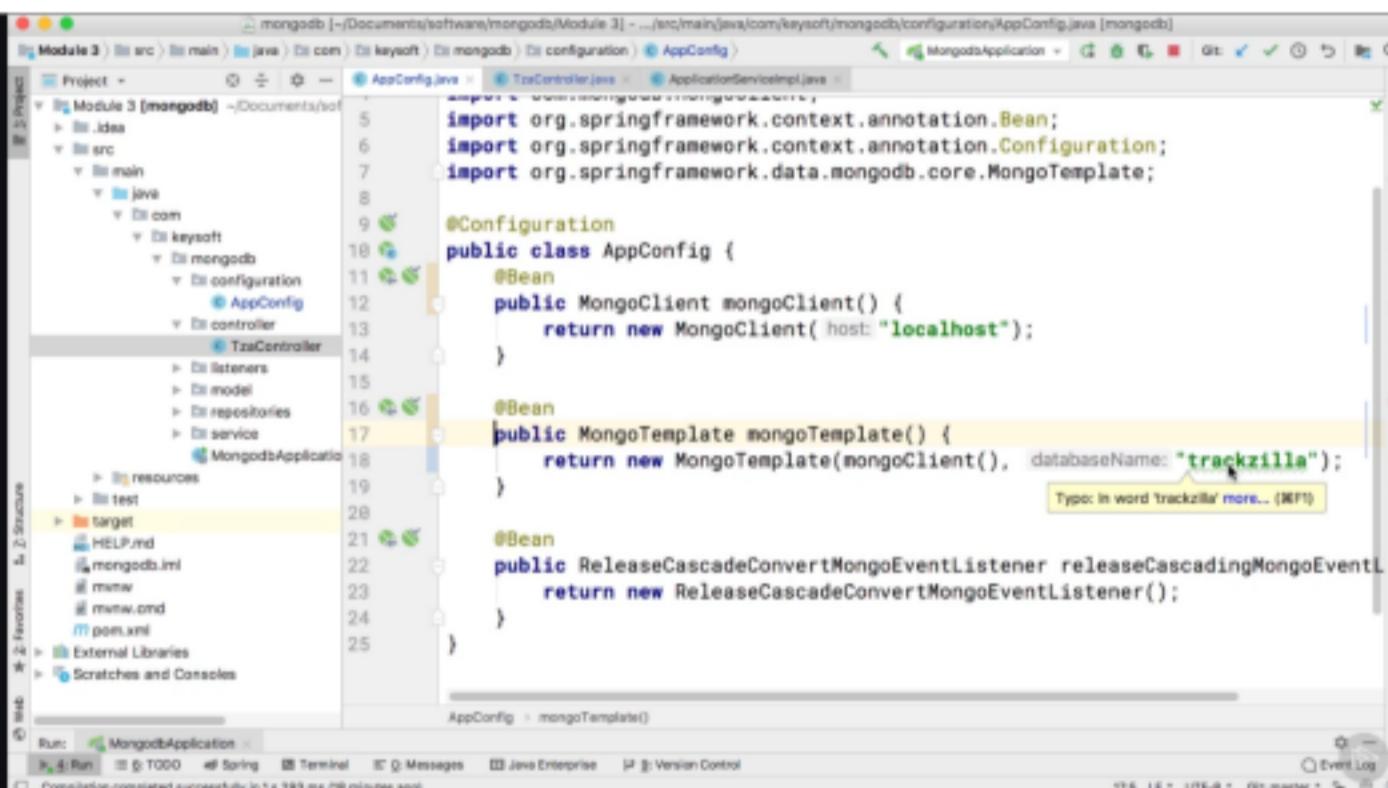
- ≡ ○ Its another way to connect to mongo db and use default methods provided by mongo template to run queries against DB.

MongoTemplate

-  Provides convenience methods
-  Operations like create, update, delete, and query
-  Mapping between domain objects and MongoDB documents
-  Provides granular control over query and results



- ≡ ○ To create Mongo Template instance we need to provide its hostname and database name.



```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.mongodb.core.MongoTemplate;

@Configuration
public class AppConfig {
    @Bean
    public MongoClient mongoClient() {
        return new MongoClient("localhost");
    }

    @Bean
    public MongoTemplate mongoTemplate() {
        return new MongoTemplate(mongoClient(), "trackzilla");
    }

    @Bean
    public ReleaseCascadeConvertMongoEventListener releaseCascadingMongoEventL
        return new ReleaseCascadeConvertMongoEventListener();
}
```

Criteria Object

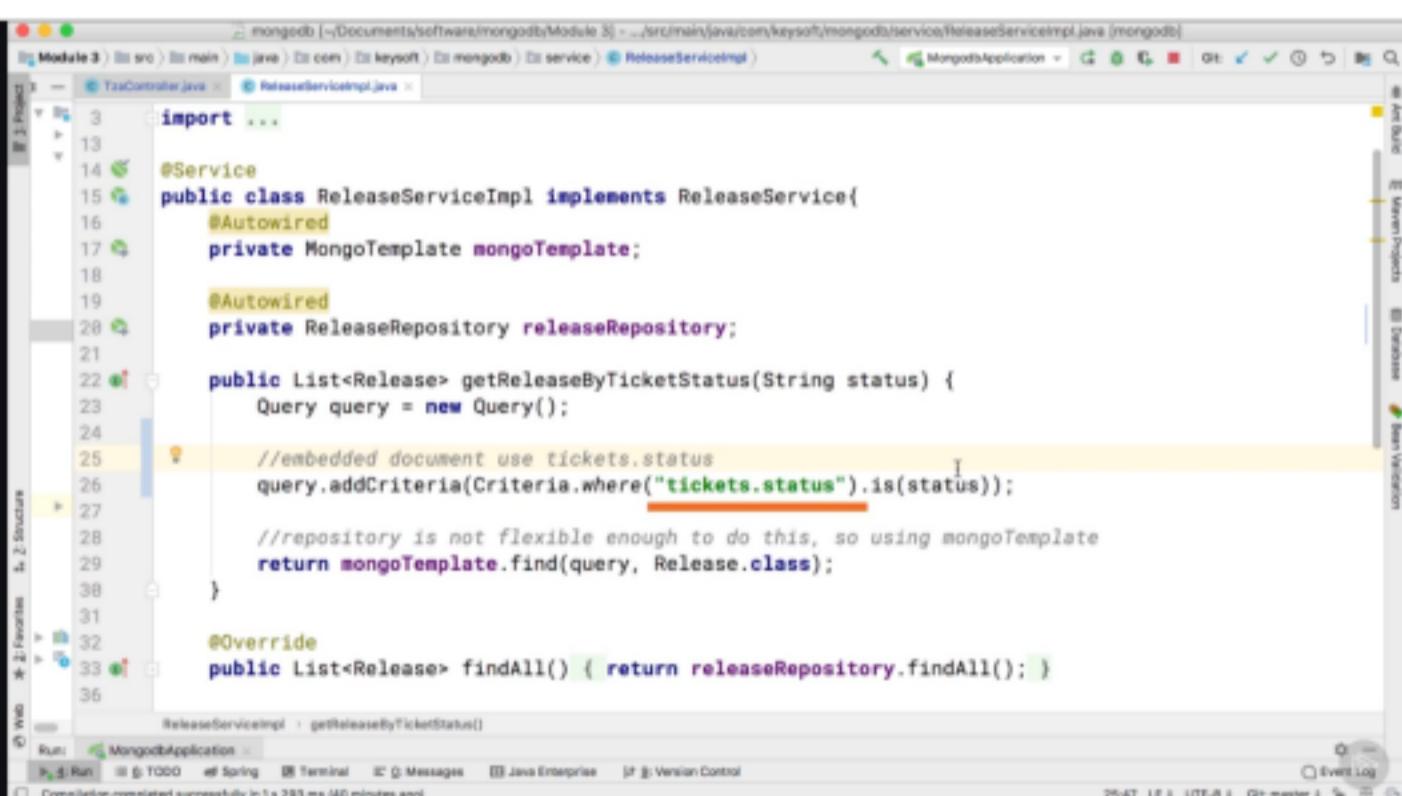
- ≡ ○ We can update mongo data by using a new method provided by Query called criteria object.
- ≡ ○ Criteria is the central class for

creating queries.

Querying embedded array of document

Release Document

```
{ "_id" : ObjectId("5ca158c74f29ee7843a00fa7"),  
  "name" : "December Release",  
  "description" : "The last release of the year.",  
  "tickets" : [ { "_id" :  
    ObjectId("5ca158c74f29ee7843a00fa5"), "title" : "Add Email  
    Notification Feature", "description" : "Email developer when  
    ticket assigned", "appId" : "5c98ea55e5a85a2435d4cbc6",  
    "status" : "Closed" }, { "_id" :  
    ObjectId("5ca158c74f29ee7843a00fa6"), "title" : "Add text  
    message notification", "description" : "Notify developer when  
    build fails", "appId" : "5c98ea55e5a85a2435d4cbc5", "status" :  
    "Closed" } ],  
  "_class" : "com.keysoft.mongodb.model.Release" }
```



The screenshot shows an IDE interface with a Java code editor. The code is part of a class named `ReleaseServiceImpl` and implements the `ReleaseService` interface. The code uses the `MongoTemplate` to query documents based on the status of their embedded `tickets` array. The relevant snippet is:

```
public List<Release> getReleaseByTicketStatus(String status) {  
    Query query = new Query();  
  
    //embedded document use tickets.status  
    query.addCriteria(Criteria.where("tickets.status").is(status));  
  
    //repository is not flexible enough to do this, so using mongoTemplate  
    return mongoTemplate.find(query, Release.class);  
}
```

Indexes in Mongo DB

- ≡ ○ Indexes help Mongo DB find documents, that match query criteria, without performing a

collection scan. If a query has appropriate index, mongo db uses the index and limits the number of documents it is examined.

- ≡ ○ This internally increases the performance of the Mongo query

Indexes

Find documents

Without performing a collection scan

Index usage

Limits the number of documents examined

- ≡ ○ There are several annotations for creating indexes.

Creating Indexes

`@Indexed`

`@TextIndexed`

`@CompoundIndex`

`@GeoSpatialIndexed`

```
@Document
```

```
public class Ticket {  
    @Indexed(name = "appId_index", direction =  
    IndexDirection.ASCENDING)  
    private String appId;  
    ...
```

```
@Indexed Annotation
```

```
db.COLLECTION_NAME.createIndex({FIELD_NAME: 1}) ←
```



Other Common Annotations

10:00 ↗

Common Annotations

@Transient

Excludes field from being persisted

@Field

Changes how field names are represented

@PersistenceConstructor

Indicates constructor to use when instantiating

@Value

Transforms a value before it is used to construct object



The screenshot shows a Java code editor in an IDE. The code is for a class named Application.java, which defines a constructor annotated with @PersistenceConstructor and several getter and setter methods for fields id, name, description, and owner.

```
private String id;
private String name;
private String description;
private String owner;

public Application() {
}

@PersistenceConstructor
public Application(String id, String name, @Value("#root.owner ?: 'Unassigned'")String owner,
    this.id = id;
    this.name = name;
    this.owner = owner;
    this.description = description;
}

public String getId() { return id; }

public void setId(String id) { this.id = id; }

public String getName() { return name; }
```

Mapping and Type Conversion

- ≡ ○ How types are mapped, to and from Mongo db documents.
Spring data mongo db supports all types that can be represented in Mongo DB internal document format. In addition to standard types, spring data mongo db provides some built in converters to map additional types.

Type Conversion

Type	Mapping Type Conversion
String	Native type conversion
double, Double, float, Float	Native type conversion
int, Integer, short, Short	Native type conversion
Date, Timestamp	Native type conversion
LocalDate	Converter
DateTime (Joda)	Converter
ZonedDateTime	Not supported



≡ ○ **Mapping Options:** We can write Custom mappings for the types that are not supported. If we utilize Mongo Convertor, it will handle the mapping of all java domain object to mongo db documents, when storing or querying these objects.

Mapping Options

	MappingMongoConverter
	SimpleMongoConverter
	Custom converter



≡ ○ **MappingMongoConvertor:** Built

in converter is not provided for ZonedDateTime class. We have to use MappingMongoConverter to provide the custom converter which can convert this java object to mongo db document format.

```
@Document  
public class Release {  
    ...  
    private ZonedDateTime releaseDate;
```

ZonedDateTime Field
Represents year, month, day, time, timezone



≡ ○ How ZonedDateTime looks in java

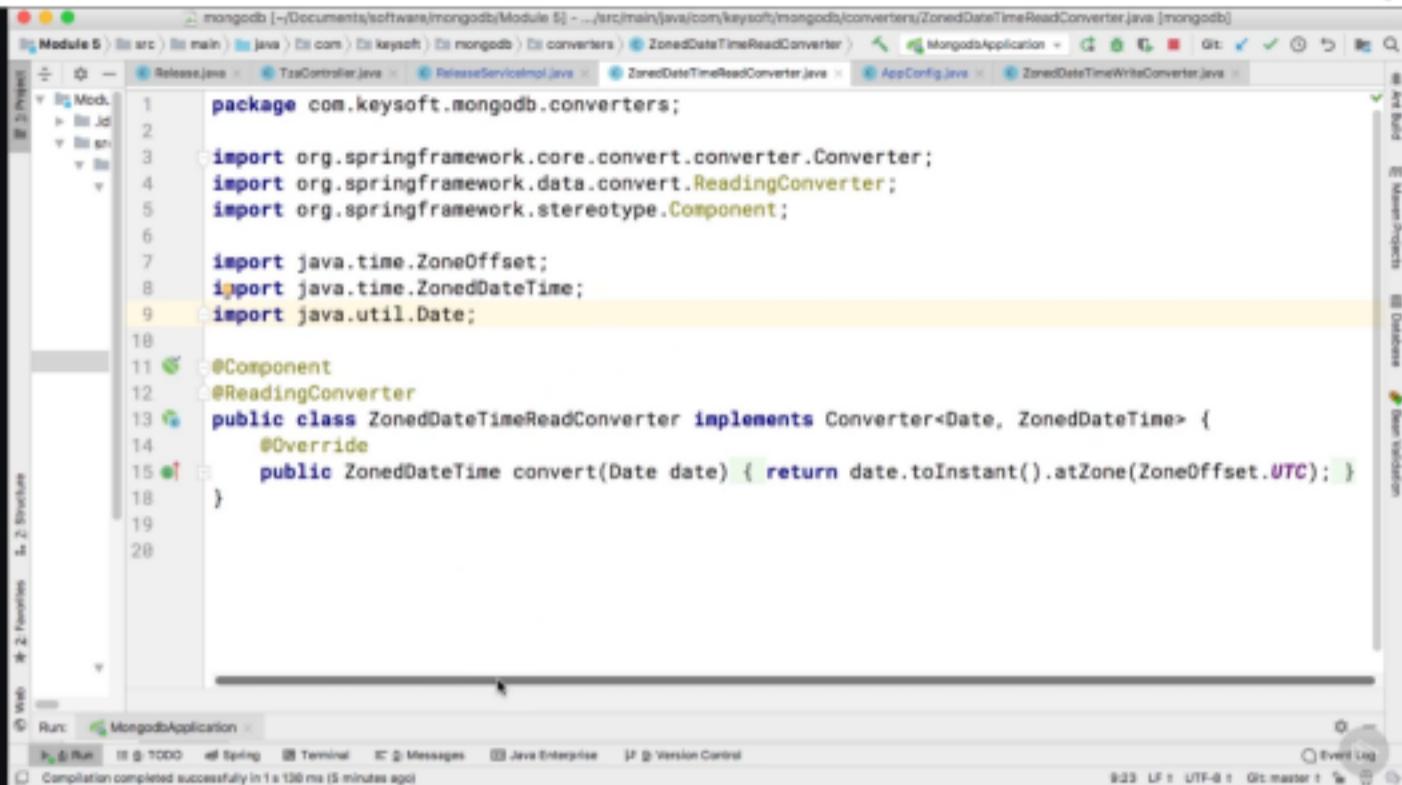
ZonedDateTime Example

```
ZonedDateTime nextFriday = LocalDateTime.now()  
    .plusHours(1)  
    .with(TemporalAdjusters.next(DayOfWeek.FRIDAY))  
    .atZone(ZoneId.of("EST"));
```



≡ ○ How to write custom convertor

for reading and writing of the type



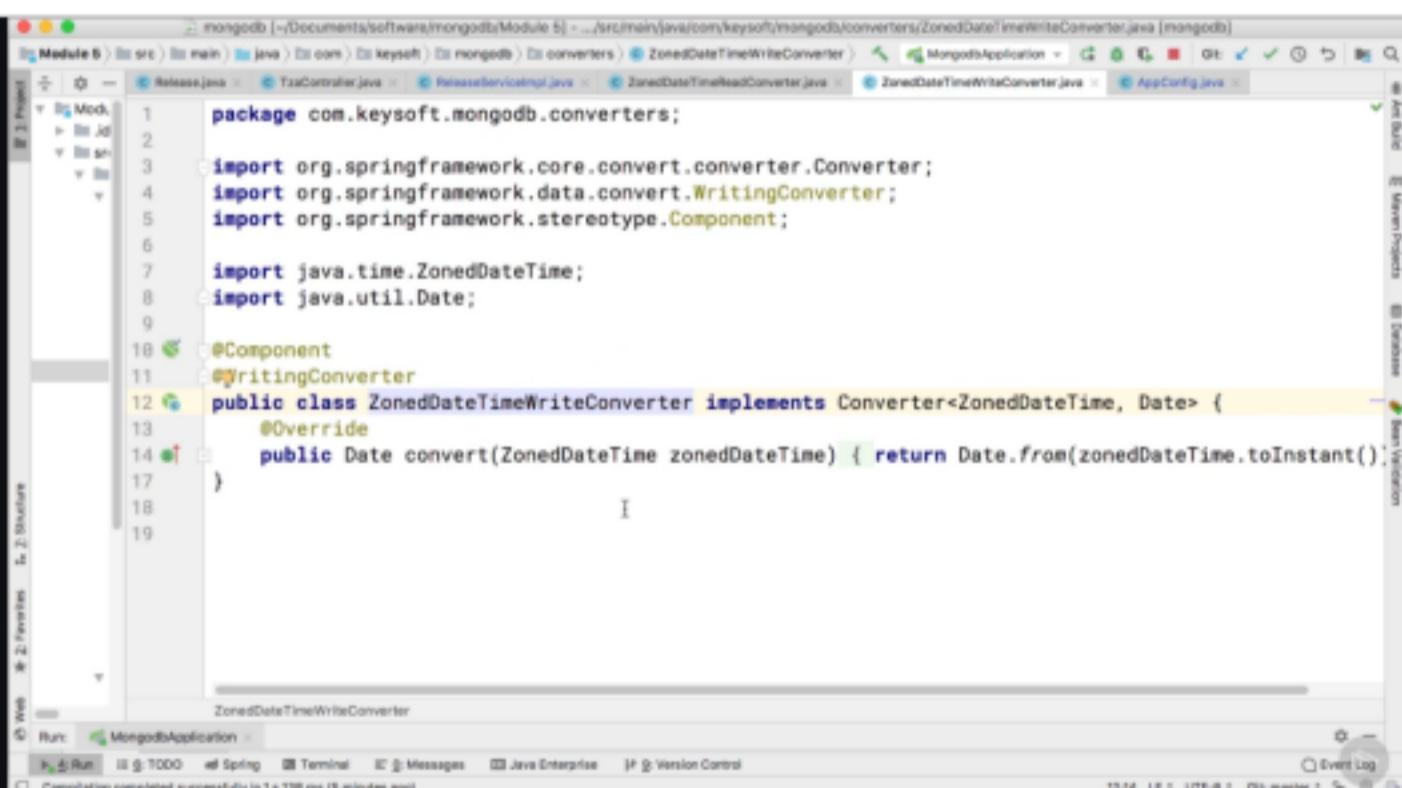
The screenshot shows a Java code editor in an IDE. The file being edited is `ZonedDateTimeReadConverter.java`. The code defines a converter that converts a `Date` object to a `ZonedDateTime` object by returning its `toInstant()` method result and then applying the `ZoneOffset.UTC`.

```
package com.keysoft.mongodb.converters;

import org.springframework.core.convert.converter.Converter;
import org.springframework.data.convert.ReadingConverter;
import org.springframework.stereotype.Component;

import java.time.ZoneOffset;
import java.time.ZonedDateTime;
import java.util.Date;

@Component
@ReadingConverter
public class ZonedDateTimeReadConverter implements Converter<Date, ZonedDateTime> {
    @Override
    public ZonedDateTime convert(Date date) { return date.toInstant().atZone(ZoneOffset.UTC); }
}
```



The screenshot shows a Java code editor in an IDE. The file being edited is `ZonedDateTimeWriteConverter.java`. The code defines a converter that converts a `ZonedDateTime` object to a `Date` object by returning its `toInstant()` method result.

```
package com.keysoft.mongodb.converters;

import org.springframework.core.convert.converter.Converter;
import org.springframework.data.convert.WritingConverter;
import org.springframework.stereotype.Component;

import java.time.ZonedDateTime;
import java.util.Date;

@Component
@WritingConverter
public class ZonedDateTimeWriteConverter implements Converter<ZonedDateTime, Date> {
    @Override
    public Date convert(ZonedDateTime zonedDateTime) { return Date.from(zonedDateTime.toInstant()); }
}
```

= ○ Finally create these custom converters and register it with MongoTemplate in a spring configuration

```
58     // converter.afterPropertiesSet();
59     // return new MongoTemplate(mongoDbFactory(), converter);
60
61     // public @Bean
62     // MongoDbFactory mongoDbFactory() {
63     //     return new SimpleMongoDbFactory(new MongoClient(), "trackzilla");
64     //
65     // }
66
67     // Bean
68     public ReleaseCascadeConvertMongoEventListener releaseCascadingMongoEventListener() {
69         return new ReleaseCascadeConvertMongoEventListener();
70     }
71
72     // Bean
73     public MongoCustomConversions customConversions() {
74         List<Converter<?, ?>> converters = new ArrayList<?>();
75         converters.add(new ZonedDateTimeReadConverter());
76         converters.add(new ZonedDateTimeWriteConverter());
77         return new MongoCustomConversions(converters);
78     }
79
80 }
```

The screenshot shows the IntelliJ IDEA interface with the code editor open to the `AppConfig.java` file. The code defines a bean for a `MongoTemplate` and another for `MongoDbFactory`. The `MongoDbFactory` bean returns a `SimpleMongoDbFactory` with a `MongoClient` and the database name `"trackzilla"`.

```
43
44
45     // Bean
46     public MongoTemplate getMongoTemplate() throws UnknownHostException {
47         MappingMongoConverter converter = new MappingMongoConverter(
48             new DefaultDbRefResolver(mongoDbFactory()), new MongoMappingContext());
49         converter.setCustomConversions(customConversions());
50         converter.afterPropertiesSet();
51         return new MongoTemplate(mongoDbFactory(), converter);
52     }
53
54     // Bean
55     public MongoDbFactory mongoDbFactory() { ←
56         return new SimpleMongoDbFactory(new MongoClient(), databaseName: "trackzilla");
57     }
58
59     // Bean
60     public ReleaseCascadeConvertMongoEventListener releaseCascadingMongoEventListener() {
61         return new ReleaseCascadeConvertMongoEventListener();
62     }
63
64     // Bean
65     public MongoCustomConversions customConversions() {
66 }
```

The screenshot shows the same `AppConfig.java` code as above, but with a red arrow pointing to the `new MongoClient()` call in the `mongoDbFactory()` method. This indicates that the `MongoClient` is being created at runtime.

Implementing Session

≡ ○ Spring Session MongoDB:

We can use `spring-session-data-mongodb` plugin to store the user session information in Mongo DB instead of in memory.

- ≡ ○ Use cases are like we can store user first name in session and later use this to display user first name in the application after login. We can also share session data between APIs
- ≡ ○ For this, first we need to create a user document with `@JsonAutoDetect` to serialize and de-serialize. Then add a configuration to create spring session bean with `@EnableMongoHttpSession`.
- ≡ ○ After this configuration there will be new collection 'sessions' created in database to store user session details.

The screenshot shows the IntelliJ IDEA interface with the User.java file open. The code defines a User class with fields for username, first name, last name, and email. It includes a constructor and a parameterized constructor. An annotation `@JsonAutoDetect(fieldVisibility = JsonAutoDetect.Visibility.ANY)` is highlighted with an orange arrow pointing to it from the left.

```
import ...  
  
@Document  
@JsonAutoDetect(fieldVisibility = JsonAutoDetect.Visibility.ANY) ←  
public class User {  
    private String userName;  
    private String firstName;  
    private String lastName;  
    private String email;  
  
    public User() {  
    }  
  
    public User(String username, String firstName, String lastName, String email) {  
        this.userName = username;  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.email = email;  
    }  
}
```

The screenshot shows the IntelliJ IDEA interface with the HttpSessionConfig.java file open. The code defines a HttpSessionConfig class with a `@Bean` method named mongoSessionConverter that returns a JacksonMongoSessionConverter. A annotation `@EnableMongoHttpSession` is highlighted with an orange arrow pointing to it from the left.

```
package com.keysoft.mongodb.configuration;  
  
import org.springframework.context.annotation.Bean;  
import org.springframework.session.data.mongo.JacksonMongoSessionConverter;  
import org.springframework.session.data.mongo.config.annotation.web.http.EnableMongoHttpSession;  
  
@EnableMongoHttpSession  
public class HttpSessionConfig {  
  
    @Bean  
    JacksonMongoSessionConverter mongoSessionConverter() {  
        return new JacksonMongoSessionConverter();  
    }  
}
```

≡ ○ We can store the user information in this Mongo DB sessions and also we can retrieve the same.

The screenshot shows a Java code editor within an IDE. The project structure on the left includes packages like com.keyssoft.mongodb, com.keyssoft.mongodb.config, com.keyssoft.mongodb.model, com.keyssoft.mongodb.repository, com.keyssoft.mongodb.service, and com.keyssoft.mongodb.util. The main file, TzaController.java, contains annotations such as @RequestMapping and @RequestBody. A specific method, getUserFromSession(), is highlighted in yellow. The code uses Spring framework annotations and MongoDB-specific types like MongoSession and Application.

```
176     @RequestMapping(value = "/releases/costs/{id}", method = RequestMethod.GET)
177     public Double getReleaseCost(@PathVariable("id") String id) { return releaseServ
178
179     //***** Methods for User Session*****
180     @RequestMapping(value = "/user", method = RequestMethod.PUT)
181     public User addUserToSession(@RequestBody User user, HttpSession session) {
182         session.setAttribute("USER", user);
183         return user;
184     }
185
186     @RequestMapping(value = "/user", method = RequestMethod.GET)
187     public Object getUserFromSession(HttpSession session){
188         return session.getAttribute("USER");
189     }
190
191     //***** Transaction Management *****
192     @RequestMapping(value = "retire/application", method = RequestMethod.DELETE)
193     public void retireApplication(@RequestBody Application application) { //just pa
194         applicationService.retireApplication(application);
195     }
196
197 }
198
199 }
```

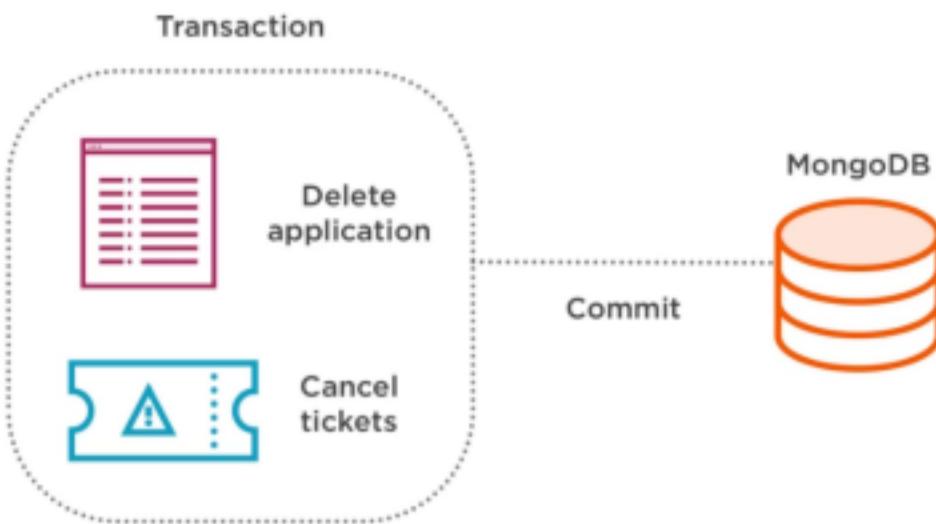
The terminal window shows a session with the mongo shell. It lists collections (application, release, sessions, ticket) and then runs a find() query on the sessions collection. The output is a single document representing a session object with fields like _id, class, createdMillis, accessedMillis, intervalSeconds, expireAt, attrs, and a nested USER object containing user details.

```
...god --config /usr/local/etc/mongod.conf
> show collections
application
release
sessions
ticket
> db.sessions.find({});
{
  "_id" : "b3874f1d-cd33-4193-a3eb-a175da4b0ac4",
  "@class" : "org.springframework.session.data.mongo.MongoSession",
  "createdMillis" : NumberLong("1554859455703"),
  "accessedMillis" : NumberLong("1554859455704"),
  "intervalSeconds" : 1800,
  "expireAt" : [
    "java.util.Date",
    NumberLong("1554861255704")
  ],
  "attrs" : {
    "@class" : "java.util.HashMap",
    "USER" : {
      "@class" : "com.keyssoft.mongodb.model.User",
      "userName" : "keshawilliams",
      "firstName" : "Kesha",
      "lastName" : "Williams",
      "email" : "keshaw@xyz.com"
    }
  },
  "principal" : null
}
>
```

Implementing Transaction

- ≡ ○ Mongo db allows transactions to run inside replica sets.

Retire an Application



- ≡ ○ **MongoTransactionManager:** Helps to implement transaction in spring mongo db projects. Create a new bean of this type in a configuration file.

MongoTransactionManager

- ➡ Manage transactions with MongoTransactionManager
- ☰ Transaction support disabled by default
- ↔ Provides easy access to Spring transaction support

```
26
27
28     @Override
29     protected String getDatabaseName() {
30         return "trackzilla";
31     }
32
33     @Bean
34     MongoTransactionManager transactionManager(MongoDbFactory dbFactory) {
35         return new MongoTransactionManager(dbFactory);
36     }
37
38     @Bean
39     public MongoDbFactory mongoDbFactory() { return new SimpleMongoDbFactory(new
40             MongoClient(), "trackzilla"); }
41
42     @Bean
43     public ReleaseCascadeConvertMongoEventListener releaseCascadingMongoEvent
44         Listener releaseCascadeConvertMongoEventListener() {
45             return new ReleaseCascadeConvertMongoEventListener();
46     }
47
48     @Bean
49     public MongoCustomConversions customConversions() {
50         List<Converter<?, ?>> converters = new ArrayList<>();
51         converters.add(new ObjectIdToHexStringConverter());
52     }
53
54 }
```

≡ ○ Then use the `@Transactional` annotation to a method which has multi step DB operations which needs to either commit(if all steps are succeeded) or rollback(if one of the step is failed).

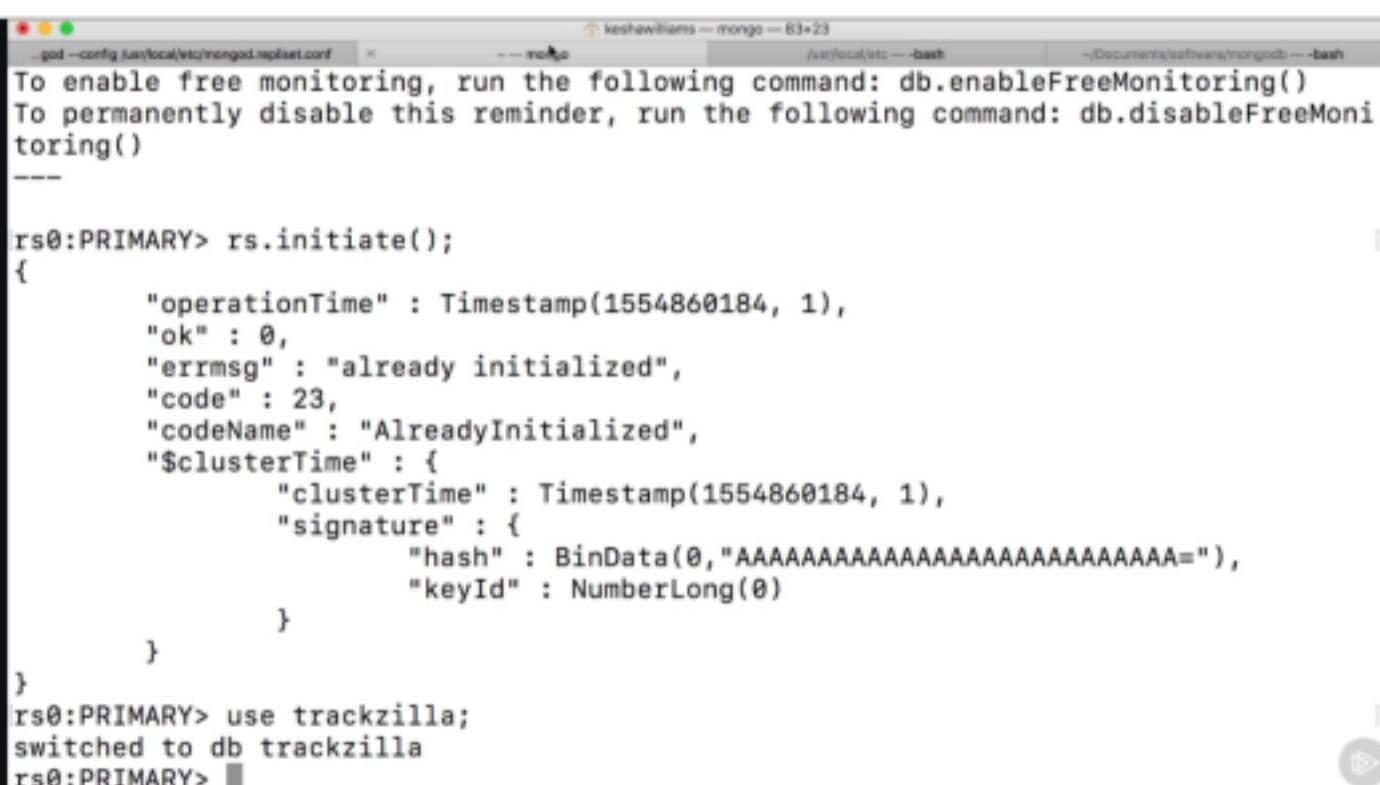
```
48
49
50     query.addCriteria(Criteria.where("name").is(application.getName()));
51     Update update = new Update();
52     update.set("name", "Trainer");
53     mongoTemplate.updateFirst(query, update, Application.class);
54
55
56     @Override
57     @Transactional
58     public void retireApplication(Application application) {
59         //Step 1
60         mongoTemplate.remove(application);
61
62         //Step 2
63         Query query = new Query();
64         query.addCriteria(Criteria.where("appId").is(application.getId()));
65         Update update = new Update();
66         update.set("status", "Cancel");
67         mongoTemplate.updateMulti(query, update, Ticket.class);
68     }
69
70 }
```

≡ ○ Note that in order to have multi step transaction replica set must

be enabled, which can be configured while running mongo db.



```
Keshas-MacBook-Pro:~ keshawilliams$ mongod --config /usr/local/etc/mongod.repliset.conf
```



```
To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
rs0:PRIMARY> rs.initiate();
{
    "operationTime" : Timestamp(1554860184, 1),
    "ok" : 0,
    "errmsg" : "already initialized",
    "code" : 23,
    "codeName" : "AlreadyInitialized",
    "$clusterTime" : {
        "clusterTime" : Timestamp(1554860184, 1),
        "signature" : {
            "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAA="),
            "keyId" : NumberLong(0)
        }
    }
}
rs0:PRIMARY> use trackzilla;
switched to db trackzilla
rs0:PRIMARY>
```

=====