Practice 6: Spring Data for MongoDb

1. In the following steps we will create a MongoDB Docker image, run a spring-boot application docker image in a separate container and link it to MongoDB Docker image.

In PowerSwell download Docker image **[1].** After downloading you should find Mongo image with docker images command.

Create a docker network boot-mongo.

```
>>  docker pull mongo

>>  docker images

>>  docker network create boot-mongo

>>  docker network ls
```

2. Instantiate the image providing a name for the container and the default MongoDB port. Create a connection to the database with MongoDB Compass.

```
>>  docker run --name mongo_awbd --network boot-mongo -p 27017:27017 mongo
```

NoSql databases

Info

- Flexible schema. Suitable for semi-structured, complex, nested data
- Does not use a structured query language, use specific query language.
- Easy to migrate.
- Typically, doesn't support (ACID) transactions.
- Ensure scalability.
- High performance.
- Support a high number of reads/sec.

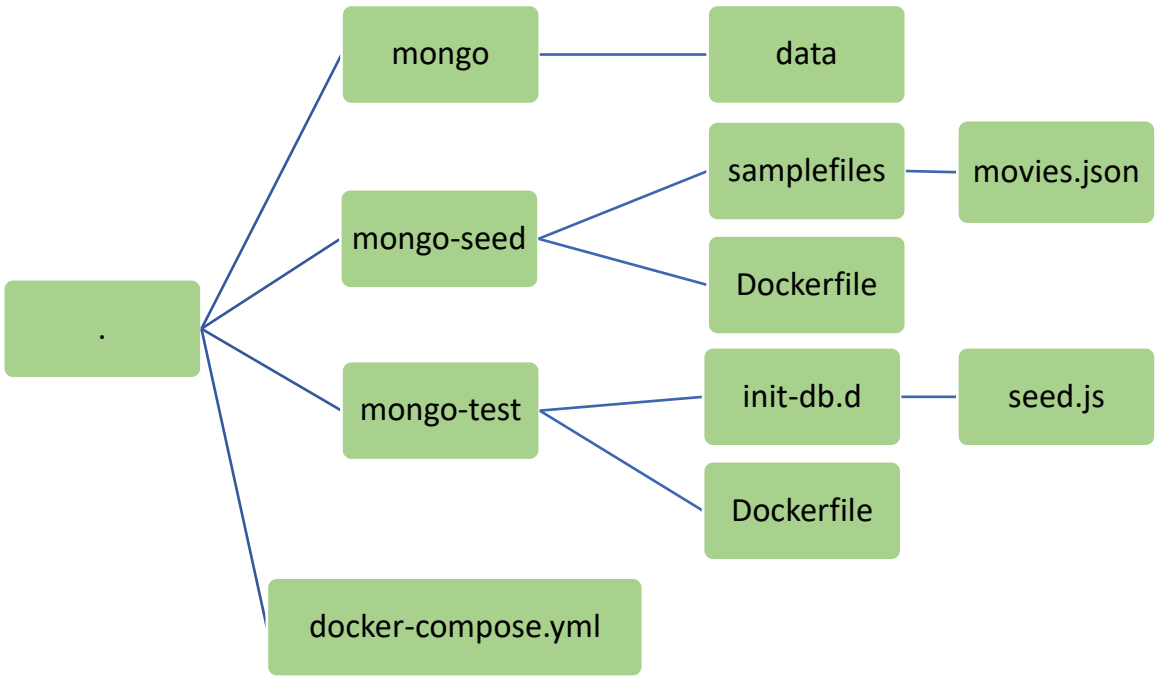
Mongo DB [2][3]

NoSql, document-oriented database.

Documents are stored as **BSON** (binary JSON) format.

Mongo	**RDBMS**
Document: set of key-value pairs, similar to JSON objects	row in a table
Collection: set of documents, documents in a collection may have different sets of fields	table
Field in JSON document	column
$lookup and embedded documents	join
Primary key	Primary key GUID

3. Create a folder with the following structure:



docker-compose.yml:

```
version: "3.5"

services:
  mongo_test:
    container_name: mongo_test
    build: ./mongo-test
    ports:
      - 27017:27017
    networks:
      - awbd
    volumes:
      - ./mongo/data:/data/db #Helps to store MongoDB data in
`./mongo/data`
    environment:
      MONGO_INITDB_ROOT_USERNAME: awbd
      MONGO_INITDB_ROOT_PASSWORD: awbd
      MONGO_INITDB_DATABASE: moviesdb

  mongo_seed:
    container_name: mongo_seed
    build: ./mongo-seed
    networks:
      - awbd
    depends_on:
      - mongo_test

networks:
  awbd:
    name: awbd
    driver: bridge
```

docker file in mongo-test

```
FROM mongo:latest
COPY ./init-db.d/seed.js /docker-entrypoint-initdb.d
```

Docker Compose [4]

Info

Docker Compose is a tool for defining and running multi-container Docker applications useful for: CI (continuous integration), CD (continuous deployment) and automated testing.

Application services are configured in a YAML file. Docker Compose caches the configuration for creating a container. To run a service that has not changed, Docker Compose re-uses the existing containers. Re-using containers speeds-up the changes made to environment.

docker-compose.yml
docker-compose.yml defines services that will run together in an isolated environment.

docker-compose up starts services.

volumes:
Compose preserves all volumes used by services. When **docker-compose up** runs, if it finds any containers from previous runs, it copies the volumes from the old container to the new container. This process ensures that any data you've created in volumes isn't lost.

build:
specifies the docker file to be run. It may be use instead of **image.**

Add in **`./init-db.d/seed.js`** commands to configure mongodb for the first use:

4.

```
db.createUser({

  user: "umovies",
  pwd: "pmovies",
  roles: [
    { role: "readWrite", db: "moviesdb" }
  ]
});
db.test.drop();
db.test.insertMany([
  {
    _id: 1,
    name: 'Ken',
    age: 40
  },
  {
    _id: 2,
    name: 'Ben',
    age: 41
  }
])
```

Check **MONGO_INITDB_DATABASE**: moviesdb, the environment variable in docker-compse.yml. We will create a user with readWrite role for this database. Also, we will create a collection, *test*, with two documents.

Both services, mongo-test and mongo-seed, will run in *awbd* network.

5. **docker** file in mongo-seed

```
FROM mongo:latest
COPY ./samplefiles/movies.json /movies.json
CMD mongoimport  --host mongo_test  --username umovies --password
pmovies --db moviesdb --collection movies --type json --jsonArray --file
/movies.json
```

We copy documents from samplefiles movies.json file and create movies collection with mongoimport **[5]** command. For more sample collections visit: **[6]**

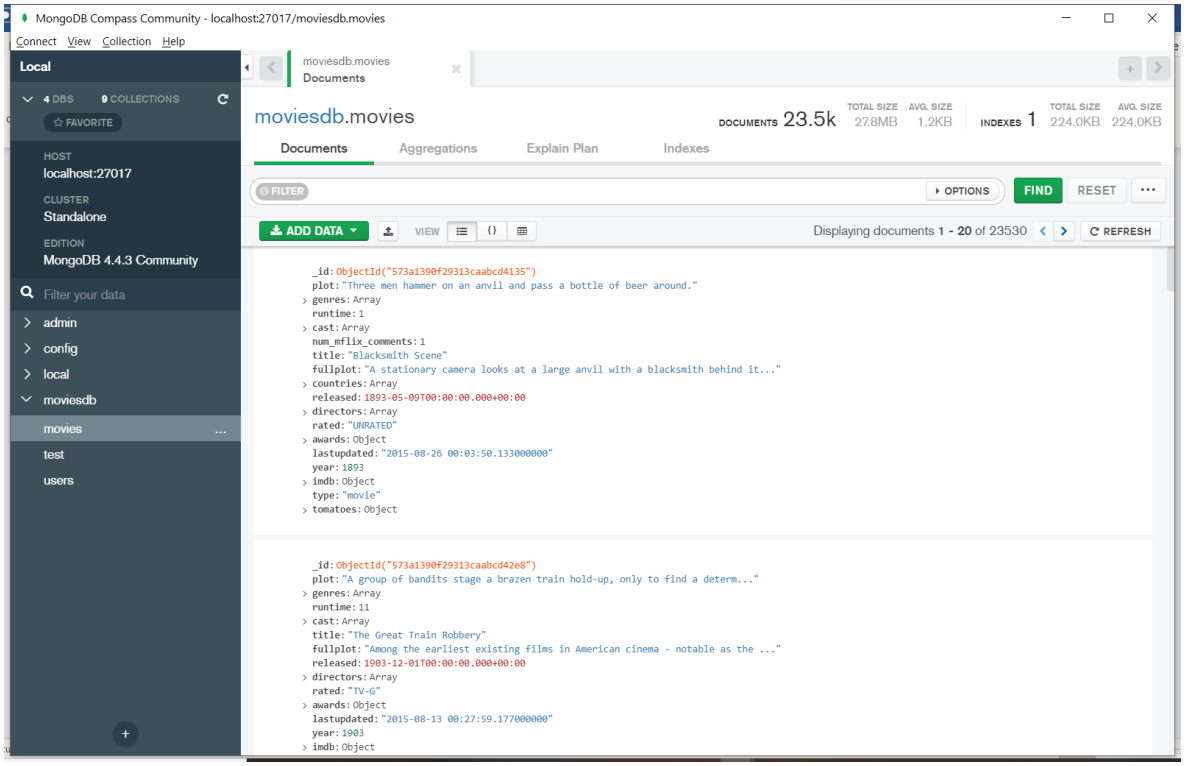
6. Run in PowerShell:

```
>>  docker-compose up
```

and connect to Mongodb with **MongoDB Compass** with the connection string:

```
mongodb://awbd:awbd@localhost:27017/moviesdb
```

Import comments.json



7. Use **Spring initializr** to generate a maven project with dependencies: Spring Data MongoDB, Lombok, Spring Web, Thymeleaf, Spring Data JPA or open LAB6_START or skip to step **10.**
https://start.spring.io/

Add dependency for bootstrap in pom.xml:

```
<dependency>
    <groupId>org.webjars</groupId>
    <artifactId>bootstrap</artifactId>
    <version>4.5.0</version>
</dependency>
```

8. Create package *com.awbd.lab7.domain* with classes Movie, Comment, Imdb:

```
@Setter
@Getter
@ToString
public class Movie {

    private String id;
    private String title;
    private String plot;
    private Date released;
    private List<String> cast;
    private String year;
    private Imdb imdb;

    private List<Comment> comments;

}
```

```
@Setter
@Getter
@ToString
public class Comment {

    private String id;

    private String name;
    private String email;
    private String text;
    private Date date;
}
```

```
@Setter
@Getter
@ToString
public class Imdb {

    private String rating;
    private String votes;

}
```

9. Create package *package com.awbd.lab6.controllers* and class *package com.awbd.lab6.controllers.IndexController*.

```
@Controller
public class IndexController {

    @RequestMapping({"", "/", "/index"})
    public String getIndexPage(){
        return "movieList";
    }

}
```

10. Annotate classes package *com.awbd.lab6.domain.Movie* and package *com.awbd.lab6.domain.Comment* with **@Document** annotation. Add @Id annotations.

```
@Setter
@Getter
@ToString
@Document(collection = "comments")
public class Comment {

    @Id
    public ObjectId id;


    private String name;
    private String email;
    private String text;
    private Date date;
}


@Document(collection = "movies")
public class Movie {
    @Id
    private ObjectId id;
//
```

Info **Mongo JPA [7][8]**

@Document is the equivalent of @Entity annotation, marking a class as domain object. We may specify the name of the collection.
For each document a file @Id must be defined.

@Filed is the equivalent of @Column annotation. Indexed fields are annotated @Indexed.

MongoRepository extends CrudRepository, providing mongo specific functionalities.

11. Check datasource configuration in *application.properties* file**.**

```
spring.data.mongodb.database=moviesdb
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
spring.data.mongodb.username=umovies
spring.data.mongodb.password=pmovies
```

12. Add package *com.awbd.lab6.repositories* and interfaces *com.awbd.lab6.repositories .CommentRepository* and *com.awbd.lab6.repositories.MovieRepository*:

```
public interface CommentRepository extends MongoRepository<Comment,
String> {

}
```

```
public interface MovieRepository extends MongoRepository<Movie, String>
{
    List<Movie> findByTitle(String title);

}
```

13. Create test package *com.awbd.lab6.repositories* and test class *com.awbd.lab6.repositories.MovieRepostioryTest*

```
@DataMongoTest
@Slf4j
public class MovieRepostioryTest {

    @Autowired
    MovieRepository movieRepository;


    @ParameterizedTest
    @ValueSource(strings = {"Civilization", "The Birth of a Nation"})
    public void findByTitle(String title) {
        List<Movie> movies = movieRepository.findByTitle(title);
        assertFalse(movies.isEmpty());
        log.info("findByTitleLike ...");
        movies.forEach(movie -> log.info(movie.toString()));
    }
}
```

Info **One to many relationships in Mongo [9][10]**

One to many may be implemented with embedded documents. To avoid repetitions, manual references or DBRef are used to link multiple documents from different collections.

DBRef include the name of the collection and the value of _id filed.

Spring Jpa annotation for DBRef is **@DBRef**.

14. Create one-to-many relationship movie-comment:

```
@DBRef(db="moviesdb")
private Set<Comment> comments = new HashSet<>();
```

15. Add autowired file *commentRepository* and test method:

```
@ParameterizedTest
@ValueSource(strings = {"573a1390f29313caabcd5a93"})
public void saveById(String id) {
    Optional<Movie> movieOpt = movieRepository.findById(id);
    assertFalse(movieOpt.isEmpty());
    log.info("findById ...");
    log.info(movieOpt.get().toString());

    Movie movie = movieOpt.get();
    Comment comment = new Comment();
    comment.setId(ObjectId.get());
    //Date date = new Date();
    //ObjectId objectIdDate = new ObjectId(date);
    comment.setText("nice movie");
    comment.setMovieId(movie.getId());
    commentRepository.save(comment);

    movie.setTitle("Civilization");
    movieRepository.save(movie);
}
```

Info

Query Methods [11][12]

Find methods in Mongo search by specific filed:value equality conditions or by specific query operators:

Examples:
db.collection_name.find({ filed:value })
db.collection_name.find({ filed: { $in: ["A", "D"] } })
db.collection_name.find({ $or: [{ filed1: "A" }, { filed2: { $lt: 30 } }] }) -- filed2 < 30 and filed1 = 'A'

SpringData JPA will automatically generate implementations for methods named according to naming conventions
*findBy**Filed**Condition*.

Alternatively, classes **Query** and **Criteria** are used to generate queries in mongo native specific manner. Both *MongoTemplate* and *MongoRepository* work with Query and Criteria.

16. Add @Query methods in class *com.awbd.lab6.repositories.MovieRepository*:

```
List<Movie> findByYearBetween(int start, int end);

@Query("{ 'year' : { $gt: ?0, $lt: ?1 } }")
List<Movie> findByYearBetweenQ(int start, int end);


@Query("{ 'title' : { $regex: ?0 } }")
List<Movie> findByTitleRegexp(String regexp);
```

17. Create test for `findByYearBetween`, `findByYearBetweenQ` and findMoviesByRegexpTitle methods:

```
@DataMongoTest
@Slf4j
public class MovieRepostioryQueryTest {

    @Autowired
    MovieRepository movieRepository;

@Test
public void findByYearBetween() {
    List<Movie> movies = movieRepository.findByYearBetween(1960,1970);
    assertFalse(movies.isEmpty());
    log.info("findByYearBetween ...");
    movies.forEach(movie -> log.info(movie.toString()));
}



@Test
public void findMoviesByRegexpTitle() {
    List<Movie> movies = movieRepository.findByTitleRegexp("^A");
    assertFalse(movies.isEmpty());
    log.info("findByRegexpTitle ...");
    movies.forEach(movie -> log.info(movie.toString()));
}

}
```

18. Create package *com.awbd.lab6.services* and classes *com.awbd.lab6.services.MovieService*, *com.awbd.lab6.services.MovieServiceImpl*:

```
public interface MovieService {
    public List<Movie> findAll();
    public Optional<Movie> findById(String id);
    public void deleteById(String id);
    Page<Movie> findPaginated(Pageable pageable);
}

@Service
public class MovieServiceImpl implements MovieService {
    @Autowired
    MovieRepository movieRepository;

    @Override
    public Optional<Movie> findById(String id) {

        return movieRepository.findById(id);
    }


    @Override
    public void deleteById(String id) {
        movieRepository.deleteById(id);
    }


    @Override
    public List<Movie> findAll() {

        return movieRepository.findAll();
    }

    @Override
    public Page<Movie> findPaginated(Pageable pageable) {
        Page<Movie> moviePage = movieRepository.findAll(pageable);
        return moviePage;
    }

}
```

19. Modify *getIndexPage* method and add autowired filed of type MovieService:

```
@RequestMapping({"", "/", "/index"})
public String getIndexPage(Model model){
    List<Movie> movies = movieService.findAll();
    model.addAttribute("movies",movies);
    System.out.println(movies.size());

    return "movieList";
}
```

20. Add *getMoviePage* method:

```
@RequestMapping({"/movies"})
public String getMoviePage(Model model,
@RequestParam("page") Optional<Integer> page,
@RequestParam("size") Optional<Integer> size) {
    int currentPage = page.orElse(1);
    int pageSize = size.orElse(10);

    Page<Movie> moviePage =
movieService.findPaginated(PageRequest.of(currentPage - 1, pageSize));

    model.addAttribute("moviePage",moviePage);

    return "moviePaginated";
}
```

21. In tymeleaf template moviePaginated.*html* modify table displaying movies to iterate the page returned by the service:

```
<tr th:each="movie : ${moviePage.content}">
....
</tr>
```

```
<li th:each="pageNumber :
${#numbers.sequence(1,T(java.lang.Math).min(7,moviePage.totalPages))}"
    th:class="${pageNumber==moviePage.number + 1} ? 'page-item active':
'page-item'">
    <a class="page-link"
       th:text="${pageNumber}"
       th:href="@{/movies(size=${moviePage.size}, page=${pageNumber})}">
        1
    </a></li>
```

22. Add class MovieController:

```
@Controller
public class MovieController {

    @Autowired
    MovieService movieService;

    @RequestMapping("/movie/info/{id}")
    public String showById(@PathVariable String id, Model model){
        Optional<Movie> movieOpt = movieService.findById(id);
        if (movieOpt.isPresent()) {
            model.addAttribute("movie", movieOpt.get());
            return "info";
        }
        else {
            model.addAttribute("id", id);
            return "nomovie";
        }
    }

    @RequestMapping("/movie/delete/{id}")
    public String deleteById(@PathVariable String id, Model model){
        Optional<Movie> movieOpt = movieService.findById(id);
        if (movieOpt.isPresent()) {
            movieService.deleteById(id);
            return "redirect:/home";
        }
        else {
            model.addAttribute("id", id);
            return "nomovie";
        }
    }

}
```

B

[1] https://hub.docker.com/_/mongo

[2] https://docs.mongodb.com/manual/introduction/

[3] https://docs.mongodb.com/manual/reference/sql-comparison/

[4] https://docs.docker.com/compose/

[5] https://docs.mongodb.com/database-tools/mongoimport/#bin.mongoimport

[6] https://docs.atlas.mongodb.com/sample-data

[7] https://docs.spring.io/spring-data/mongodb/docs/current/reference/html/#reference

[8] https://www.baeldung.com/spring-data-mongodb-tutorial

[9] https://docs.mongodb.com/manual/tutorial/model-referenced-one-to-many-relationships-between-documents/

[10] https://docs.mongodb.com/manual/reference/database-references/

[11] https://www.baeldung.com/queries-in-spring-data-mongodb

[12] https://docs.mongodb.com/manual/tutorial/query-documents/

[13] https://www.baeldung.com/spring-thymeleaf-pagination