Passos realizados na aula de Spring – Parte 3 – MongoDB

Aula 01

Parte 1 – Primeiro contato com o MongoDB

Instalar o MongoDB Community Server
- Colocar o mongodb no PATH do Sistema Operacional.
- Definir um diretório para ele armazenar os dados: ex: mongo/data
- Testar no terminal: mongod

Entrar no Mongo e criar a base de dados `aula-01` e a coleção `Messages`,

Inserir alguns documentos `Message` manualmente.

Parte 2 – Ajustando o projeto Spring Boot para acessar o MongoDB

Incluir a dependência do MongoDB no pom.xml:

```
<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
```

Criar a interface MessageMongoRepository:

```
@Repository
public interface MessageMongoRepository extends MongoRepository<Message, Integer> {}
```

Atualizar a entity `Message` para usar as anotações do Mongo: `@Document` e `@Id` (coleção do MongoDB). Incluir o nome do documento no `@Document`
- Incluir um atributo `@Id private String id; //com set e get`
  - Pacote do @Id: `org.springframework.data.annotation.Id`
- Remover as anotações do JPA (pacote `javax.persistence`)

Criar uma classe de service: `MessageMongoService` com o método `findAll`.

Substituir o `MessageService` por `MessageMongoService` no controller.

Comentar o código de acesso ao `MessageRepository` (antigo) na classe `Aula01Application`.

Incluir os dados de conexão com o MondoDB no arquivo application.properties.
`spring.data.mongodb.uri=mongodb://localhost:27017/<nome_do_banco>`

Testar o endpoint `/messages`, usando GET.

Aula 02 (usando objetos aninhados)

Criar a classe User no pacote Model:

```
@Document(collection="Users")
public class User implements Serializable {
	@Id
	private String id;
	private String name;
	private String email;
	//Criar construtor default, construtor com parâmetros (sem o id), setters e getters
```

Criar uma interface `UserMongoRepository`.

```
@Repository
public interface UserMongoRepository extends MongoRepository<User, String> {}
```

Atualizar a classe `Messages` para incluir o `User`.

private User user; //criar set e get

Limpar o documento da `aula-01` no MongoDB.

Criar o documento de `User` no MongoDB.

Editar o runner (main) para criar e salvar alguns objetos no MongoDB.

Injetar os repositories.

```
@Autowired
private UserMongoRepository userRepo;

@Autowired
private MessageMongoRepository msgMongoRepo;
```

Criar os objetos de usuário:

```
User u1 = new User("User 001");
User u2 = new User("User 002");
User u3 = new User("User 003");
```

Remover e salvar os usuários:

```
userRepo.deleteAll();
userRepo.saveAll(Arrays.asList(u1, u2, u3));
```

Criar os objetos de mensagem:

```
Message m1 = new Message("1", "Primeira mensagem",
MessagePriority.MEDIUN);
Message m2 = new Message("2", "Segunda mensagem",
```

```
MessagePriority.HIGH);
```

Setar os usuários nas mensagens:

```
m1.setUser(u2);
m2.setUser(u3);
```

Salvar as mensagens daquele usuário. Antes de salvar, você pode chamar o método `deleteAll()` do repository para limpar o banco.

```
msgMongoRepo.deleteAll();
msgMongoRepo.saveAll(Arrays.asList(m1, m2));
```

Executar o projeto e verificar os objetos armazenados no MongoDB.

Aula 03 (usando referências)

Adicionar a seguinte anotação no atributo `User` da classe `Message`:

```
@DBRef(lazy = true)
private User user;
```

Executar o projeto e verificar os dados armazenados no MongoDB.

Perceba a diferença de usar agregado e usar referências.

Exercício: discuta com o colega sobre a importância desta diferença e cite exemplos onde cada estratégia é melhor para ser aplicada.

Aula 04 (extra)

Criar uma classe de UserDTO, incluindo as mensagens nele.
Testar o GET, POST, PUT and DELETE.