A continuación se presenta una guía paso a paso para replicar y ejecutar el proyecto en un ejemplo de aplicación **Spring Boot** con integración a **MongoDB**. Cada paso incluye los detalles específicos, herramientas y configuraciones necesarias, desde la instalación de MongoDB hasta la creación de endpoints y pruebas mediante Swagger.

1. Preparar el Entorno

1. **Instalar Java**
    - Asegúrate de tener instalada una versión compatible de Java (el proyecto usa *JDK 11*).
    - Verifica la instalación con el comando: `java -version`
    -
2. **Instalar Docker** (si deseas ejecutar MongoDB en contenedor local)
    - Descarga e instala Docker de acuerdo a tu sistema operativo (Windows, Mac o Linux).
    - Inicia Docker Desktop (o el equivalente en tu SO) antes de continuar.
3. **Opcional: MongoDB en la Nube**
    - Si no quieres usar Docker, puedes crear una cuenta en MongoDB Atlas y configurar una base de datos en la nube.
    - Obtén la cadena de conexión y credenciales proporcionadas por Atlas para usarlas en el archivo de configuración de Spring Boot.

2. Configurar y Ejecutar MongoDB (Local con Docker)

1. **Descargar Imagen de Mongo**
    - Abre una terminal y ejecuta: `docker pull mongo`
    - Esto descargará la imagen oficial de MongoDB.
2. **Crear y Ejecutar Contenedor**
    - En la misma terminal, ejecuta un comando similar al siguiente para levantar el contenedor de Mongo: `docker run -d \`
    - `--name mongoDB \`
    - `-e MONGO_INITDB_ROOT_USERNAME=admin \`
    - `-e MONGO_INITDB_ROOT_PASSWORD=password \`
    - `-p 27017:27017 \`
    - `mongo`
    -

        - `--name mongoDB`: Nombre del contenedor.
        - `-e MONGO_INITDB_ROOT_USERNAME` / `-e MONGO_INITDB_ROOT_PASSWORD`: Usuario y contraseña de administración.
        - `-p 27017:27017`: Mapea el puerto local 27017 al contenedor.
3. **Verificar el Contenedor**
    - Usa `docker ps` para comprobar que el contenedor "mongoDB" está en ejecución.
    - (Opcional) Instala Studio 3T u otra herramienta GUI para conectarte y visualizar las colecciones dentro de MongoDB.

3. Crear el Proyecto Spring Boot

1. **Usar Spring Initializr (opción recomendada)**
    - Visita start.spring.io.
    - Configura el proyecto con:
        - **Grupo (GroupId)**: `com.ejemplo` (o el que desees).
        - **Artefacto (ArtifactId)**: `spring-boot-mongodb` (por ejemplo).
        - **Lombok**: Añade la dependencia para minimizar código boilerplate.
        - **Spring Web**: Para crear controladores REST.
        - **Spring Data MongoDB**: Para la integración con MongoDB.
        - **Versiones**: Asegúrate de usar Java 11 o la versión adecuada.
2. **Usar IntelliJ** (opción alternativa)
    - Crea un nuevo proyecto **Maven** o **Gradle** con compatibilidad para Spring Boot.
    - Agrega manualmente en el `pom.xml` o `build.gradle` las dependencias de:
        - `spring-boot-starter-web`
        - `spring-boot-starter-data-mongodb`
        - `lombok`
3. **Estructura de Paquetes**
    - Organiza el código de la siguiente manera (recomendación típica): `src/main/java`

```
    └─ ... (paquetes base)
        ├─ collection  (Clases que representan colecciones de Mongo)
        ├─ controller  (Controladores REST)
        ├─ repository  (Interfaces para MongoRepository)
        ├─ service     (Lógica de negocio)
        └─ config      (Configuraciones adicionales, ej. Swagger)
```

4. Configurar la Conexión con MongoDB

1. **Archivo de Propiedades**

    - Ve a `src/main/resources/application.properties` (o `.yml` si prefieres YAML).
    - Define las propiedades de conexión a MongoDB, por ejemplo:
    - `spring.data.mongodb.host=localhost`
    - `spring.data.mongodb.port=27017`
    - `spring.data.mongodb.database=demo`
    - `spring.data.mongodb.username=admin`
    - `spring.data.mongodb.password=password`
    - `spring.data.mongodb.authentication-database=admin`
    -
    - `# Opcional, si gestionas el tamaño de subida de ficheros`
    - `spring.servlet.multipart.max-file-size=256MB`
    - `spring.servlet.multipart.max-request-size=256MB`
    -
    - Ajusta credenciales, host y puerto según tu contenedor/local o MongoDB Atlas (en caso de uso en la nube, reemplaza host/puerto/credenciales por tu cadena de conexión).
2. **Verificar la Conexión**

    - Al iniciar la aplicación de Spring Boot, deberías ver en el log que se conecta correctamente al contenedor de MongoDB (o a la instancia en la nube).

5. Crear las Colecciones (Entidades)

5.1. Entidad `Person`

1. **Clase `Person`**
    - Dentro del paquete `collection`, crea una clase `Person.java` con los campos relevantes: `@Document(collection = "person")`
    - `@Data`
    - `@Builder`
    - `@JsonInclude(JsonInclude.Include.NON_NULL)`
    - `public class Person {`
    - `    @Id`
    - `    private String personId;`
    - `    private String firstName;`
    - `    private String lastName;`
    - `    private Integer age;`

```
    private List<String> hobbies;
    private List<Address> addresses;
}
```

2. **Clase `Address`**
   - También en `collection`, crea la clase `Address.java`: `@Data`
   - `@Builder`
   - `public class Address {`
   - `    private String address1;`
   - `    private String address2;`
   - `    private String city;`
   - `}`
   - `Person` contendrá una lista de `Address` para manejar múltiples domicilios.

5.2. Repositorio `PersonRepository`

1. **Interfaz `PersonRepository`**
   - En el paquete `repository`, crea: `@Repository`
   - `public interface PersonRepository extends MongoRepository<Person, String> {`
   - `    // Ejemplo de método de búsqueda personalizado`
   - `    List<Person> findByFirstNameStartsWith(String name);`
   - `}`
   - Extiende `MongoRepository` indicando el tipo de la entidad (`Person`) y el tipo de ID (`String`).

5.3. Servicio `PersonService`

1. **Interfaz y Clase de Implementación**
   - Crea una interfaz `PersonService` y luego `PersonServiceImpl` en el paquete `service`: `public interface PersonService {`
   - `    String save(Person person);`
   - `    List<Person> getPersonStartsWith(String name);`
   - `    void delete(String id);`
   - `    // ... otros métodos ...`
   - `}`
   - `@Service`

```
public class PersonServiceImpl implements
PersonService {
    @Autowired
    private PersonRepository personRepository;

    @Override
    public String save(Person person) {
        personRepository.save(person);
        return person.getPersonId();
    }

    @Override
    public List<Person> getPersonStartsWith(String
name) {
        return
personRepository.findByFirstNameStartsWith(name);
    }

    @Override
    public void delete(String id) {
        personRepository.deleteById(id);
    }
    // ... implementación adicional ...
}
```

5.4. Controlador `PersonController`

1. **Crear Endpoints**
   - En el paquete `controller`, crea la clase `PersonController`:

```
@RestController
@RequestMapping("/person")
public class PersonController {

    @Autowired
    private PersonService personService;

    @PostMapping
    public String save(@RequestBody Person person)
{
        return personService.save(person);
```

```
        }

        @GetMapping
        public List<Person>
    getPersonStartsWith(@RequestParam String name) {
            return
    personService.getPersonStartsWith(name);
        }

        @DeleteMapping("/{id}")
        public void delete(@PathVariable String id) {
            personService.delete(id);
        }
        // ... otros endpoints ...
    }
```

- Asegúrate de ajustar las rutas y métodos HTTP según tu preferencia.

6. Integrar Swagger para Documentación y Pruebas (Utilizar la versión 2.7.2. de SpringBoot)

1. **Dependencias en `pom.xml`** `<dependency>`
2. `    <groupId>io.springfox</groupId>`
3. `    <artifactId>springfox-boot-starter</artifactId>`
4. `    <version>3.0.0</version>`
5. `</dependency>`
6. `<dependency>`
7. `    <groupId>io.springfox</groupId>`
8. `    <artifactId>springfox-swagger-ui</artifactId>`
9. `    <version>3.0.0</version>`
10. `</dependency>`
11.
12. **Configuración de Swagger**
    - Crea una clase de configuración en `config/SpringFoxConfig.java`:

```
    @Configuration
    public class SpringFoxConfig {
        @Bean
        public Docket api() {
            return new
    Docket(DocumentationType.SWAGGER_2)
```

```
                    .select()
                    .apis(RequestHandlerSelectors.any(
))
                    .paths(PathSelectors.any())
                    .build();
    }
}
```

- En la clase principal de tu aplicación (`SpringBootMongodbApplication`), habilita: `@EnableSwagger2`

```
@SpringBootApplication
public class SpringBootMongodbApplication { ... }

Añadir en el archivo application.properties:
spring.mvc.pathmatch.matching-strategy =
ANT_PATH_MATCHER
```

13. **Probar en el Navegador**
    - Inicia la aplicación y visita: `http://localhost:8080/swagger-ui/`
    - Deberías ver la documentación generada automáticamente y poder probar los endpoints (`/person`, etc.) desde allí.

7. Probando la Aplicación

1. **Iniciar el Proyecto**
    - Ejecuta la clase principal (por ejemplo, `SpringBootMongodbApplication.java`) desde tu IDE o mediante Maven/Gradle: `mvn spring-boot:run`
2. **Verificar la Conexión**
    - Observa la consola para comprobar que la aplicación se haya conectado a MongoDB.
    - Asegúrate de que no haya mensajes de error relacionados con credenciales o puertos.
3. **Consumir Endpoints**
    - Desde **Swagger UI** (por defecto, `http://localhost:8080/swagger-ui/`) o herramientas como **Postman**:
        - **POST /person**: Envía un JSON para crear un `Person`.

        - **GET /person?name={texto}**: Filtra personas cuyo `firstName` comience con el texto proporcionado.
        - **DELETE /person/{id}**: Elimina un `Person` según el `id`.
4. **Verificar en MongoDB**
    - En **Studio 3T** (o cualquier GUI):
        - Abre la conexión `localhost:27017`, autenticándote con `admin` / `password` (o las credenciales que hayas configurado).
        - Busca la base de datos `demo` y la colección `person`.
        - Verifica que los documentos reflejen las inserciones y eliminaciones realizadas.

8. Uso de MongoTemplate y Operaciones Avanzadas (Opcional)

1. **Consultas Personalizadas**
    - Para búsquedas o agregaciones complejas, en el servicio puedes inyectar `MongoTemplate` y construir consultas con `Criteria` y `Query`.
    - Ejemplo básico: `@Autowired`
    - `private MongoTemplate mongoTemplate;`
    -
    - `public List<Person> findByAgeRange(int minAge, int maxAge) {`
    - `    Query query = new Query();`
    - `query.addCriteria(Criteria.where("age").gte(minAge).lte(maxAge));`
    - `    return mongoTemplate.find(query, Person.class);`
    - `}`
    -
2. **Agregaciones**
    - Para operaciones como `group by`, `sort`, `unwind`, etc.: `Aggregation agg = Aggregation.newAggregation(`
    - `    Aggregation.unwind("addresses"),`
    - `    Aggregation.sort(Sort.Direction.DESC, "age"),`
    - `    Aggregation.group("addresses.city").first("$$ROOT").as("oldestPerson")`
    - `);`

- `List<Document> result = mongoTemplate.aggregate(agg, Person.class, Document.class).getMappedResults();`
-
- Estas agregaciones permiten manipular datos en formato BSON/Document a alto nivel.

9. Manejo de Archivos Binarios (Opcional)

1. **Entidad `Photo`**
   - Crear nueva clase `Photo` en `collection`: `@Document(collection = "photo")`
   - `@Data`
   - `@JsonInclude(JsonInclude.Include.NON_NULL)`
   - `public class Photo {`
   - `    @Id`
   - `    private String id;`
   - `    private String title;`
   - `    private Binary photo;`
   - `}`
   -
2. **Repositorio y Servicio**
   - Crear `PhotoRepository` extendiendo `MongoRepository<Photo, String>` y un `PhotoService`.
   - Implementar métodos para guardar (`save`) la imagen como `Binary`.
3. **Subida/Descarga de Imágenes**
   - Utilizar `MultipartFile` para recibir el archivo en el controlador y luego guardarlo en MongoDB.
   - Para descargar, recuperar el `Binary` y retornarlo con `ResponseEntity<Resource>`.

10. Verificación Final y Consejos

1. **Verificar que todos los endpoints funcionen**
   - Comprueba la ejecución de cada uno de los métodos (GET, POST, DELETE).
   - Observa los logs de la aplicación por posibles excepciones.
2. **Agregar Manejo de Excepciones**

    - Es recomendable incluir un controlador de excepciones global (`@ControllerAdvice`) para manejar errores y responder con códigos HTTP apropiados.
3. **Personalizar Configuraciones**
    - Ajusta puertos (si 8080 está ocupado).
    - Ajusta la estructura de carpetas y nombres de paquetes a tus preferencias.
    - Cambia las credenciales de MongoDB por valores más seguros en producción.
4. **Uso en Producción**
    - Para entornos productivos, considera un archivo `application-prod.properties`, así como la configuración de logs y monitorización (Spring Actuator, etc.).

Conclusión

Siguiendo estos pasos podrás:

- **Configurar** un proyecto Spring Boot básico.
- **Integrar** MongoDB localmente o a través de MongoDB Atlas.
- **Crear** colecciones y repositorios para realizar operaciones de CRUD.
- **Exponer** endpoints REST y documentarlos con Swagger.
- **Probar** el correcto funcionamiento mediante Swagger UI, Postman o cualquier otra herramienta de tu preferencia.
- **Realizar** agregaciones y operaciones avanzadas con `MongoTemplate`.
- **Manejar** almacenamiento de archivos binarios (por ejemplo, imágenes) dentro de MongoDB.

Con esta guía, tienes los conocimientos fundamentales en Java y Spring Boot podrá replicar el proyecto descrito en el texto adjunto y ejecutarlo de principio a fin.