Add Three Test Cases

Testcases

My testcases are focused on `MongoProperties` class. Originally, this class is covered in this status:

Element	Class, %	Method, %	Line, %
embedded	100% (6/6)	100% (26/26)	96% (86/89)
MongoAutoConfiguration	100% (1/1)	100% (1/1)	100% (2/2)
MongoClientFactory	100% (1/1)	100% (11/11)	100% (43/43)
MongoProperties	100% (1/1)	81% (18/22)	83% (30/36)
MongoReactiveAutoConfiguration	100% (3/3)	100% (7/7)	100% (22/22)
ReactiveMongoClientFactory	100% (1/1)	100% (15/15)	100% (50/50)

We planned to cover all methods in `MongoProperties`. Our testcases are presented as follows:

1. Test if database could be accessed.

   This test is to test `getDatabase()` method. Normally, we only need to set the property and get the property with this method. But, an interesting thing is that spring-boot doesn't write in that way. From other testcases, I learned that we could build a `MongoClient` and initialize it with our properties, then use `getCredentialsList()` to get the credentials of the database. With this process, we could make sure that the flow of generating and accessing database works fine.

```
1  @Test
2      void databaseCanBeAccessed(){
3          // Build a database
4          MongoProperties properties = new MongoProperties();
5          // Build a test database
6          MongoProperties test = new MongoProperties();
7          properties.setDatabase("W2020");
8          properties.setUsername("user");
9          properties.setPassword("secret".toCharArray());
10         test.setDatabase(properties.getDatabase());
11         test.setUsername(properties.getUsername());
12         test.setPassword(properties.getPassword());
13         MongoClient clientTest = createMongoClient(test);
14         assertMongoCredential(getCredentials(clientTest).get(0),
   "user", "secret", "W2020");
15     }
```

2. Test Set GridFs Module

   FsDatabase is a special type of MongoDB that stores larger files. The setter and getter function are not covered in the original tests. Following the design pattern of other testcases, we design this testcase to test if we could set FsDatabase and get this database.

```
@Test
    void gridFsDatabaseCanBeSet(){
        MongoProperties properties = new MongoProperties();
        properties.setGridFsDatabase("W2020");
        properties.setUsername("user");
        properties.setPassword("secret".toCharArray());
        MongoClient clientTest = createMongoClient(properties);
        assertFsCredential(getCredentials(clientTest).get(0), "user",
"secret", "W2020");
    }
```

3. Test get GridFs Module.

   This test helps to decide if we could set up a `MongoClient` with FridFsDatabase and get its properties by methods of `MongoClient`. The test logic follows other testcases written by spring-boot team.

```
@Test
    void gridFsDatabaseCanBeAccessed(){
        MongoProperties properties = new MongoProperties();
        MongoProperties test = new MongoProperties();
        properties.setGridFsDatabase("W2020");
        properties.setUsername("user");
        properties.setPassword("secret".toCharArray());
        test.setGridFsDatabase(properties.getGridFsDatabase());
        test.setUsername(properties.getUsername());
        test.setPassword(properties.getPassword());
        MongoClient clientTest = createMongoClient(test);
        assertFsCredential(getCredentials(clientTest).get(0), "user",
"secret", "W2020");
    }
```

4. Test `setFieldNamingStrategy`.

   This function helps to configure the `FieldNamingStrategy` to be used to determine the field name of MongoDB if no manual mapping is applied. `FieldNamingStrategy` is an interface which decides how to name documents fields in cases field name is not manually defined. To simplify test process, I just wrote a dummy class to simulate the `FieldNamingStrategy`. A really interesting thing a learned is that my dummy class is `TestHelper`, but `getName()` function actually return the name with package name. This is a new stuff I learned!

```
@Test
    void setStrategy(){
        MongoProperties test = new MongoProperties();
        test.setFieldNamingStrategy(TestHelper.class);

Assert.assertEquals("org.springframework.boot.autoconfigure.mongo.MyMong
oTestCase$TestHelper", test.getFieldNamingStrategy().getName());
    }
```

The current coverage is as below:

Element	Class, %	Method, %	Line, %
embedded	100% (6/6)	100% (26/26)	96% (86/89)
MongoAutoConfiguration	100% (1/1)	100% (1/1)	100% (2/2)
MongoClientFactory	100% (1/1)	100% (11/11)	100% (43/43)
MongoProperties	100% (1/1)	100% (22/22)	100% (36/36)
MongoReactiveAutoConfiguration	100% (3/3)	100% (7/7)	100% (22/22)
ReactiveMongoClientFactory	100% (1/1)	100% (15/15)	100% (50/50)

To make tests work, we use these helper functions as well.

```
1  private void assertMongoCredential(MongoCredential credentials, String
   expectedUsername, String expectedPassword,
2                                     String expectedSource) {
3          assertThat(credentials.getUserName()).isEqualTo(expectedUsername);
4
   assertThat(credentials.getPassword()).isEqualTo(expectedPassword.toCharArra
   y());
5          assertThat(credentials.getSource()).isEqualTo(expectedSource);
6      }
7
8      private void assertFsCredential(MongoCredential credentials, String
   expectedGridFsUsername, String expectedPassword,
9                                      String expectedSource) {
10
   assertThat(credentials.getGridFsName()).isEqualTo(expectedGridFsUsername);
11
   assertThat(credentials.getPassword()).isEqualTo(expectedPassword.toCharArra
   y());
12         assertThat(credentials.getSource()).isEqualTo(expectedSource);
13     }
14
15     private MongoClient createMongoClient(MongoProperties properties) {
16         return createMongoClient(properties, null);
17     }
18
19     private MongoClient createMongoClient(MongoProperties properties,
   Environment environment) {
20         return new MongoClientFactory(properties,
   environment).createMongoClient(null);
21     }
22
23     private List<MongoCredential> getCredentials(MongoClient client) {
24         return client.getCredentialsList();
25     }
```