Kundera / src / kundera-mongo / src / test / java / com / impetus / client / crud / gfs / **GridFSTest.java**

devender-yadav #665 92cf521 on May 7 2015

1 contributor

366 lines (313 sloc) 9.75 KB Raw Blame History

```
/*******************************************************************************
 *  * Copyright 2015 Impetus Infotech.
 *  *
 *  * Licensed under the Apache License, Version 2.0 (the "License");
 *  * you may not use this file except in compliance with the License.
 *  * You may obtain a copy of the License at
 *  *
 *  *      http://www.apache.org/licenses/LICENSE-2.0
 *  *
 *  * Unless required by applicable law or agreed to in writing, software
 *  * distributed under the License is distributed on an "AS IS" BASIS,
 *  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 *  * See the License for the specific language governing permissions and
 *  * limitations under the License.
 ******************************************************************************/
package com.impetus.client.crud.gfs;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.Query;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Assert;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

import com.impetus.client.utils.MongoUtils;

/**
 * The Class GridFSTest.
 *
 * @author Devender Yadav
 */
public class GridFSTest
{

    /** The Constant _PU. */
    private static final String _PU = "gfs_pu";

    /** The emf. */
    private static EntityManagerFactory emf;

    /** The em. */
    private EntityManager em;

    /** The profile pic1. */
    private byte[] profilePic1;

    /** The profile pic2. */
    private byte[] profilePic2;

    /** The profile pic3. */
```

```java
    private byte[] profilePic3;

    /**
     * Sets the up before class.
     *
     * @throws Exception
     *             the exception
     */
    @BeforeClass
    public static void SetUpBeforeClass() throws Exception
    {
        emf = Persistence.createEntityManagerFactory(_PU);
    }

    /**
     * Sets the up.
     *
     * @throws Exception
     *             the exception
     */
    @Before
    public void setUp() throws Exception
    {
        em = emf.createEntityManager();
    }

    /**
     * Tear down.
     *
     * @throws Exception
     *             the exception
     */
    @After
    public void tearDown() throws Exception
    {
        em.close();
    }

    /**
     * Tear down after class.
     *
     * @throws Exception
     *             the exception
     */
    @AfterClass
    public static void tearDownAfterClass() throws Exception
    {
        MongoUtils.dropDatabase(emf, _PU);
        emf.close();
    }

    /**
     * Test CRUD GridFS.
     *
     * @throws Exception
     *             the exception
     */
     @Test
    public void testCRUDGridFS() throws Exception
    {
        testInsert();
        testUpdateNonLobField();
        testUpdateLobField();
        testDelete();
    }

    /**
     * Test query.
     *
     * @throws Exception
     *             the exception
     */
    @Test
    public void testQuery() throws Exception
    {
        profilePic1 = createBinaryData("src/test/resources/pic.jpg", 10);
        GFSUser user1 = prepareUserObject(1, "Dev", profilePic1);

        profilePic2 = createBinaryData("src/test/resources/pic.jpg", 15);
        GFSUser user2 = prepareUserObject(2, "PG", profilePic2);

        profilePic3 = createBinaryData("src/test/resources/pic.jpg", 20);
        GFSUser user3 = prepareUserObject(3, "Amit", profilePic3);

        em.persist(user1);
        em.persist(user2);
        em.persist(user3);

        em.clear();

        String query = "SELECT u FROM GFSUser u";
        Query qry = em.createQuery(query);
        List<GFSUser> userList = qry.getResultList();
        Assert.assertEquals(3, userList.size());
        assertUsers(userList, true, true, true);

        query = "SELECT u FROM GFSUser u WHERE u.name = 'Dev'";
        qry = em.createQuery(query);
        userList = qry.getResultList();
        GFSUser user = userList.get(0);
        Assert.assertEquals("Dev", user.getName());
        Assert.assertEquals(profilePic1.length, user.getProfilePic().length);

        query = "SELECT u FROM GFSUser u WHERE u.name = 'Karthik'";
        qry = em.createQuery(query);
        userList = qry.getResultList();
        Assert.assertEquals(true, userList.isEmpty());

        query = "SELECT u FROM GFSUser u WHERE u.userId = 2";
        qry = em.createQuery(query);
        userList = qry.getResultList();
        user = userList.get(0);
        Assert.assertEquals("PG", user.getName());
        Assert.assertEquals(profilePic2.length, user.getProfilePic().length);

        query = "SELECT u FROM GFSUser u WHERE u.userId > 1";
        qry = em.createQuery(query);
        userList = qry.getResultList();
        Assert.assertEquals(2, userList.size());
        assertUsers(userList, false, true, true);

        query = "SELECT u FROM GFSUser u WHERE u.userId = 2 and u.name = 'PG'";
        qry = em.createQuery(query);
        userList = qry.getResultList();
        Assert.assertEquals(1, userList.size());
        user = userList.get(0);
        Assert.assertEquals("PG", user.getName());
        Assert.assertEquals(profilePic2.length, user.getProfilePic().length);

        query = "SELECT u FROM GFSUser u WHERE u.userId < 3 order by u.name DESC";
        qry = em.createQuery(query);
        userList = qry.getResultList();
        Assert.assertEquals(2, userList.size());
        assertUsers(userList, true, true, false);

        // remove all users
        GFSUser u1 = em.find(GFSUser.class, 1);
        GFSUser u2 = em.find(GFSUser.class, 2);
        GFSUser u3 = em.find(GFSUser.class, 3);
        em.clear();
        em.remove(u1);
        em.remove(u2);
        em.remove(u3);
    }

    /**
     * Test insert.
     */
    private void testInsert()
    {
        byte[] profilePic = createBinaryData("src/test/resources/pic.jpg", 20);
        GFSUser user = prepareUserObject(1, "Dev", profilePic);
        em.persist(user);

        em.clear();

        GFSUser u = em.find(GFSUser.class, 1);
        Assert.assertEquals("Dev", u.getName());
        Assert.assertEquals(profilePic.length, u.getProfilePic().length);
    }

    /**
     * Test update non lob field.
     */
    private void testUpdateNonLobField()
    {
        GFSUser user = em.find(GFSUser.class, 1);
        user.setName("Devender");
        em.merge(user);

        em.clear();

        GFSUser u1 = em.find(GFSUser.class, 1);

        Assert.assertNotNull(u1);
        Assert.assertEquals("Devender", u1.getName());
    }

    /**
     * Test update lob field.
     */
    private void testUpdateLobField()
    {
        GFSUser user = em.find(GFSUser.class, 1);
        byte[] profilePic = createBinaryData("src/test/resources/pic.jpg", 15);
        user.setProfilePic(profilePic);

        em.merge(user);

        em.clear();

        GFSUser u = em.find(GFSUser.class, 1);

        Assert.assertNotNull(u);
        Assert.assertEquals("Devender", u.getName());
        Assert.assertEquals(profilePic.length, u.getProfilePic().length);
    }

    /**
     * Test delete.
     */
    private void testDelete()
    {
        GFSUser user = em.find(GFSUser.class, 1);
        em.remove(user);
        em.clear();
        GFSUser u = em.find(GFSUser.class, 1);
        Assert.assertNull(u);
    }

    /**
     * Prepare user object.
     *
     * @param userID
     *            the user id
     * @param name
     *            the name
     * @param profilePic
     *            the profile pic
     * @return the GFS user
     */
    private GFSUser prepareUserObject(int userID, String name, byte[] profilePic)
    {
        GFSUser user = new GFSUser();
        user.setUserId(userID);
        user.setName(name);
        user.setProfilePic(profilePic);
        return user;
    }

    /**
     * Creates the binary data.
     *
     * @param locationPath
     *            the location path
     * @param multiplicationFactor
     *            the multiplication factor
     * @return the byte[]
     */
    private byte[] createBinaryData(String locationPath, int multiplicationFactor)
    {
        Path path = Paths.get(locationPath);
        byte[] data = null;
        try
        {
                data = Files.readAllBytes(path);
            }
            catch (IOException e)
            {
                Assert.fail();
            }

            byte[] multipliedData = new byte[data.length * multiplicationFactor];

            for (int i = 0; i < multiplicationFactor; i++)
            {
                System.arraycopy(data, 0, multipliedData, i * data.length, data.length);
            }

            return multipliedData;
        }

        /**
         * Assert users.
         *
         * @param userList
         *            the user list
         * @param foundUser1
         *            the found user1
         * @param foundUser2
         *            the found user2
         * @param foundUser3
         *            the found user3
         */
        private void assertUsers(List<GFSUser> userList, boolean foundUser1, boolean foundUser2, boolean foundUser3)
        {
            for (GFSUser user : userList)
            {
                if (user.getUserId() == 1)
                {
                    Assert.assertEquals("Dev", user.getName());
                    Assert.assertEquals(profilePic1.length, user.getProfilePic().length);
                }

                else if (user.getUserId() == 2)
                {
                    Assert.assertEquals("PG", user.getName());
                    Assert.assertEquals(profilePic2.length, user.getProfilePic().length);
                }

                else if (user.getUserId() == 3)
                {
                    Assert.assertEquals("Amit", user.getName());
                    Assert.assertEquals(profilePic3.length, user.getProfilePic().length);
                }

                else
                {
                    Assert.fail();
                }
            }
        }

    }
```