TESTING ALBUM ADMIN WEBSITE

Group 6

Names	:	Faza Naziila Dias Shanti	(2320010208)
		Salvia Maylie Budiawati	(2320010276)
Class	:	3SE4	
Faculty	:	Mr. Riza Muhammad Nurman, S. Kom.	

CEP CCIT

FAKULTAS TEKNIK UNIVERSITAS INDONESIA

2024

PROJECT ON

Testing Album Admin Website

Developed by

1. **Faza Naziila Dias Shanti (232001020)**
2. **Salvia Maylie Budiawati (2320010276)**

Testing Album Admin Website

Batch Code : 3SE4

Start Date : October 3rd, 2024

End Date : October 23rd, 2024

Name of Faculty : Riza Muhammad Nurman, S.Kom

Names of Developer :

1. Faza Naziila Dias Shanti (2320010208)
2. Salvia Maylie Budiawati (2320010276)

Date of Submission: October 24th, 2024

NIIT

CERTIFICATE

This is to certify that this report titled "Testing Album Admin Website" embodies the original work done by Faza Naziila Dias Shanti and Salvia Maylie Budiawati. Project in partial fulfillment of their course requirement at NIIT.

Coordinator:

Riza Muhammad Nurman, S.Kom

ACKNOWLEDGEMENT

Praise and gratitude to Almighty God for the completion of the project entitled "Testing Album Admin Website" as a prerequisite for continuing study at CCIT-FTUI. For the moral and material support provided in the preparation of this paper, the authors expresses their gratitude to:

1. Dr. Ir. Muhammad Suryanegara, ST, M.Sc, IPM. as CCIT-FTUI director who has given guidance, advice, idea and also the opportunity to use the school facilities to support the creation of project.
2. Riza Muhammad Nurman, S.Kom is our mentor lecturer who gives encouragement, and input to the authors.
3. The parents of the authors who gave a lot of support both morally and materially.
4. All the parties cannot be mentioned in detail one by one who have helped in the process of drafting this project.

The authors realize that this project is far from a perfect word and still has some shortcomings, therefore the authors very much expect constructive advice and criticism from the reader for the perfection of this project.

SYSTEM ANALYSIS

System Summary:

Album Admin website is designed to provide a platform for users to archives NCT's data. The website allows users to view the information of NCT sub-units and the NCT members' profiles, including images and roles, presented in a structured and responsive format. Also, the website enable users to do Create, Read, Update, and Delete (CRUD) the data. The website is built using JavaServer Faces (JSF) for the user interface, Hibernate ORM for database management, GlassFish as the application server, bootstrap for responsive design, and MySQL to store the data.

Album Admin website obligated user to sign in with username and password that has stored beforehand to access the website. It is also allowed users to Create, Read, Update, and Delete data. The fitures are functionatae across the website. The Create, Read, Update, and Delete available in admin page can be used to manipulate Album, News, and Member data. The Album Admin website has six pages that can be accessed through navigation bar, those are: Admin page, News page, NCT 127 page, NCT DREAM page, NCT U page, and WAY V page.

As the website has done, as a programmer, there is no guarantee that there are no human mistake had made. Hence, to make the website into a perfect one, the testing with Junit testing using Java Netbeans, Blackbox testing for manual test, JaCoCo, and JMeter for load testing should be done. We tested the methods in the DAO and model.

SYSTEM ANALYSIS

Test Planning

The test planning of the black-box testing

index.xhtml

No.	Test Case	Test Data	Expected Result
1.	user Sign in by clicking Sign In button	User click Sign In button	go to signin.xhtml page

signin.xhtml

No.	Test Case	Test Data	Expected Result
1	user input username and password into the Sign In form and click the Sign In button	User fill the valid username and password	the system will go to the admin page and display the username on the top section of the page
2	user input username and password into the Sign In form and click the submit button	User fill the wrong username and password	the system will show "Login Failed"
3	User click the cancel button in the Sign In page	Use click the cancel button	the system will display the index page

SYSTEM ANALYSIS

admin.xhtml

No.	Test Case	Test Data	Expected Result
1	user add a new album by filling the content	User input the content data: album image, album title, description, album links, and unit ID	stay in admin page and the album data will be updated, and the inputed data will be saved in the database
2	user add a new album with empty content data	The content left empty	stay in admin page and the album data will be updated, and the data will be stay unchanged
3	user edit the data of an album and click save button	User change the album data	stay in admin page and the album data will be updated, and the edited data will be saved in the database
4	user delete an album by clicking the delete button	user delete an album	stay in admin page and the album data will be updated, and the chosen data will be deleted in the database
5	user add a new news row by filling news data and click the Add News button	User input the content data: news image, news content, Unit ID	stay in admin page and the news data will be updated, and the inputed data will be saved in the database. The inputed data will be showed in the News page
6	user add a new news row by left the news data empty and click the Add News button	The content left empty	stay in admin page and the news data will be updated, and the data will be stay unchanged
7	user edit either the news image or news content and then click the save button	User change a news image/news content	stay in admin page and the news data will be updated, and the edited data will be saved in the database. The data will be updated in the News page

SYSTEM ANALYSIS

admin.xhtml

No.	Test Case	Test Data	Expected Result
8	user delete a news by clicking the delete button	user delete a news	stay in admin page and the news data will be updated, the chosen data will be deleted in the database and in the News page.
9	user add a new member by inputing the data and click add new member button	user input the content data: member name, image address, role, and unit	stay in admin page with "Success" message and the data will be updated both in the page and in the database. The data will be shown in the each Unit's page
10	user add a new member by left the data empty and click add new member button	The content left empty	stay in admin page and the news data will be updated, and the data will be stay unchanged
11	user edit a member's name, image address, role and click the save button	user change a member's name, image address, or role, or all of it	stay in admin page with "Success" message and the data will be updated both in the admin page and in the database. The data also will be updated in the each Unit's page
12	user delete a member by clicking delete button	user delete a member	stay in admin page with "Success" message and the data will be updated both in the admin page and in the database. The data also will be deleted in the each Unit's page
13	user logging out b by clicking Log Out button in the admin.xhtml page	user click Log Out button	go to index.xhtml page

SYSTEM ANALYSIS

header.xhtml

No.	Test Case	Test Data	Expected Result
1	go to news.xhtml page by click News text on the navigation bar	click News text on the navigation bar	display news.xhtml page
2	go to nct127.xhtml page by click NCT 127 text on the navigation bar	click NCT 127 text on the navigation bar	display nct127.xhtml page
3	go to nctDream.xhtml page by click NCT Dream text on the navigation bar	click NCT Dream text on the navigation bar	display nctDream.xhtml page
4	go to nctU.xhtml page by click NCT U text on the navigation bar	click NCT U text on the navigation bar	display nctU.xhtml page
5	go to wayV.xhtml page by click WayV text on the navigation bar	click WayV text on the navigation bar	display wayV.xhtml page
6	go to admin.xhtml page by click Admin text on the navigation bar	click Admin text on the navigation bar	display admin.xhtml page

CONFIGURATION

Hardware : Lenovo

Operating System : Windows 10

Software : NetBeans IDE 8.2, XAMPP, Microsoft Office Word, Microsoft Power Point

PROJECT FILE DETAILS

	File Name	Remarks
1	PROJECT 2 Group 6 Testing Admin Website	Microsoft word containing the report
2	testing Album admin website.pptx	Microsoft Power Point containing the presentation slide show
3	nct.rar	NetBeans file
4	nct1.sql	Database

TYPE OF TESTING TOOLS

- **Junit**

  JUnit is an open-source testing framework provided for Java programming language. It allows developers to write and run repeatable tests, making sure the code works as e xpected. This framework provides various annotations to test the code and makes the testing process simple and quick.

- **JaCoCo**

  JaCoCo is a free code coverage libarary for Java. It helps measure how much of the code is being tested by the unit tests, pointing out areas that might need more testing. The plugin colors all java files according to the unit test coverage information.

- **Mockito**

  Mockito is an open-source, testing framework that you can use along with JUnit to unit test Java applications. Mockito allows us to verify the behaviour of the system under test without establishing expectations beforehand. It is also allowing us to create mock objects to simulate the behaviour of real objects. This way, we can isolate the piece of code we are testing, making sure any external dependencies do not interfere.

- **Black Box**

  Black-box testing is a method where the tester evaluates the functionality of an application without looking at its internal structures or workings. Creating tests based on requirements and user scenarios without looking at the actual code. This means we can discover issues related to user experience, data handling, or integration that might not be obvious from just looking at the code.

TYPE OF TESTING TOOLS

- **JMeter**

  JMeter is an open-source tool used for performance testing and load testing of applications. It can simulate multiple users sending requests to a server, web application, or service, and measures how the system performs under different conditions.

TESTING IMPLEMENTATION

1. AdminDAOTest

```
@Test
public void testGetBy() {
    System.out.println("testGetBy");

    // Pengujian untuk login yang berhasil
    List<Tbladmin> result = instance.getBy("admin1", "admin123");
    assertNotNull(result); // Memastikan hasil tidak null
    assertFalse(result.isEmpty()); // Memastikan hasil tidak kosong

    // Pengujian untuk login yang gagal
    List<Tbladmin> errorResult = instance.getBy("admin", "wrongpassword");
    assertNotNull(errorResult); // Memastikan hasil tidak null
    assertTrue(errorResult.isEmpty()); // Memastikan hasil kosong

    // Pengujian untuk username yang tidak ada
    List<Tbladmin> noUserResult = instance.getBy("non_existing_user", "any_password");
    assertNotNull(noUserResult); // Memastikan hasil tidak null
    assertTrue(noUserResult.isEmpty()); // Memastikan hasil kosong
}
```

1.1 TestGetBy()

Tests the getBy method of the AdminDAO class.

- Successful login: Tests with valid credentials ("admin1", "admin123") to ensure that the method returns a non-null and non-empty result.
- Failed login: Tests with incorrect credentials ("admin", "wrongpassword") to ensure that the method returns a non-null but empty result.
- Non-existing user: Tests with a non-existent username ("non_existing_user", "any_password") to ensure that the method returns a non-null and empty result.

TESTING IMPLEMENTATION

2. AlbumDAOTest.java

```
@Test
public void testAddAlbum() {
    // Menggunakan album yang sama
    Tblalbum retrievedAlbum = albumDAO.getAlbumById(testAlbum.getAlbumId());
    assertNotNull(retrievedAlbum); // Pastikan album tidak null
    assertEquals("Test Album", retrievedAlbum.getAlbumName()); // Pastikan nama album sesuai
}
```

```
@Test
public void testUpdateAlbum() {
    testAlbum.setAlbumName("Updated Album");
    albumDAO.updateAlbum(testAlbum);
    Tblalbum updatedAlbum = albumDAO.getAlbumById(testAlbum.getAlbumId());
    assertEquals("Updated Album", updatedAlbum.getAlbumName()); // Pastikan nama album diperbarui
}
```

```
@Test
public void testDeleteAlbum() {
    albumDAO.deleteAlbum(testAlbum);
    Tblalbum deletedAlbum = albumDAO.getAlbumById(testAlbum.getAlbumId());
    assertNull(deletedAlbum); // Pastikan album sudah dihapus
}
```

```
@Test
public void testGetAlbumById() {
    Tblalbum album = albumDAO.getAlbumById(testAlbum.getAlbumId());
    assertNotNull(album); // Pastikan album tidak null
    assertEquals("Test Album", album.getAlbumName()); // Pastikan nama album sesuai
}
```

```
@Test
public void testGetAllAlbums() {
    // Mengambil semua album dari database
    List<Tblalbum> albums = albumDAO.getAllAlbums();

    // Memastikan daftar album tidak null
    assertNotNull(albums);

    // Memastikan ada album dalam daftar
    assertFalse(albums.isEmpty());

    // Memastikan album yang ditest ada dalam daftar
    boolean albumExists = albums.stream().anyMatch(album -> album.getAlbumId().equals(testAlbum.getAlbumId()));
    assertTrue(albumExists); // Pastikan album yang ditest ada dalam daftar
}
```

TESTING IMPLEMENTATION

```
@Test
public void testGetAlbumById() {
    Tblalbum album = albumDAO.getAlbumById(testAlbum.getAlbumId());
    assertNotNull(album); // Pastikan album tidak null
    assertEquals("Test Album", album.getAlbumName()); // Pastikan nama album sesuai
}
```

```
@Test
public void testGetAlbumsByUnitId() {
    List<Tblalbum> albums = albumDAO.getAlbumsByUnitId("U1");
    assertNotNull(albums); // Pastikan daftar album tidak null
    assertFalse(albums.isEmpty()); // Pastikan ada album dalam daftar
    assertEquals("U1", albums.get(0).getTblunit().getUnitId()); // Pastikan unit ID sesuai
}
```

```
@Test
public void testUpdateAlbumWithInvalidId() {
    Tblalbum nonExistentAlbum = new Tblalbum();
    nonExistentAlbum.setAlbumId(-1); // ID yang tidak ada
    nonExistentAlbum.setAlbumName("Non-existent Album");
    albumDAO.updateAlbum(nonExistentAlbum);
    Tblalbum result = albumDAO.getAlbumById(-1);
    assertNull(result); // Pastikan album tidak ada
}
```

2.1 testAddAlbum()

Tests the addAlbum method of AlbumDAO by ensuring that the album inserted during the setUp() method is properly added to the database. It checks whether the album is retrievable by its ID and verifies the album's name.

2.2 testUpdateAlbum()

Tests the updateAlbum method of AlbumDAO by modifying the name of testAlbum. The method then retrieves the updated album from the database and checks whether the name has been updated.

2.3 testDeleteAlbum()

This method tests the deleteAlbum method of AlbumDAO. It adds a new album, deletes it, and verifies that the album can no longer be retrieved from the database. The test asserts that the deleted album is null.

TESTING IMPLEMENTATION

2.4 testGetAllAlbums()

This method tests the getAllAlbums method of AlbumDAO. It retrieves a list of all albums from the database and asserts that the list is not null or empty.

2.5 testGetAlbumById()

This method tests the getAlbumById method of AlbumDAO. It retrieves the testAlbum and verifying that it is not null and its name matches the expected value.

2.6 testGetAlbumsByUnitId()

This method tests the getAlbumsByUnitId method of AlbumDAO. It retrieves albums by a specific unit ID and asserts that the list is not null or empty. It also checks that all retrieved albums belong to the expected unit ID.

2.7 testUpdateAlbumWithInvalidId()

Function: Tests the updateAlbum method with an invalid album ID (-1). It ensures that updating a non-existent album does not affect the database and checks that the album with ID -1 does not exist (returns null).

TESTING IMPLEMENTATION

3. MemberDAOTest.java

```
@Test
public void testAddMember() {
    memberDAO.addMember(testMember);
    Tblmember retrievedMember = memberDAO.getMemberById("M1");
    assertNotNull(retrievedMember);
    assertEquals(testMember.getMemberName(), retrievedMember.getMemberName());
}
```

```
@Test
public void testUpdateMember() {
    memberDAO.addMember(testMember);
    testMember.setMemberName("Updated Name");
    memberDAO.updateMember(testMember);
    Tblmember updatedMember = memberDAO.getMemberById("M1");
    assertEquals("Updated Name", updatedMember.getMemberName());
}
```

```
@Test
public void testDeleteMember() {
    memberDAO.addMember(testMember);
    memberDAO.deleteMember(testMember);
    Tblmember deletedMember = memberDAO.getMemberById("M1");
    assertNull(deletedMember);
}
```

```
@Test
public void testGetMemberById() {
    memberDAO.addMember(testMember);
    Tblmember retrievedMember = memberDAO.getMemberById("M1");
    assertNotNull(retrievedMember);
    assertEquals(testMember.getMemberId(), retrievedMember.getMemberId());
}
```

```
@Test
public void testGetAllMembers() {
    memberDAO.addMember(testMember);
    List<Tblmember> members = memberDAO.getAllMembers();
    assertNotNull(members);
    assertFalse(members.isEmpty());
}
```

```
@Test
public void testGetMembersByUnitId() {
    memberDAO.addMember(testMember);
    List<Tblmember> members = memberDAO.getMembersByUnitId("U1");
    assertNotNull(members);
    assertFalse(members.isEmpty());
    assertEquals("U1", members.get(0).getTblunit().getUnitId());
}
```

TESTING IMPLEMENTATION

3.1 testAddMember()

Tests the addMember method by adding testMember to the database. It then retrieves the member by its ID ("M1") and verifies that the member exists (assertNotNull) and its name matches the expected name.

3.2 testUpdateMember()

Tests the updateMember method by adding testMember first, then updating its name to "Updated Name". It retrieves the member and verifies that the name has been correctly updated.

3.3 testDeleteMember()

Tests the deleteMember method by adding testMember first, then deleting it. After deletion, it verifies that the member no longer exists in the database by retrieving it and asserting that the result is null.

3.4 testGetMemberById()

Tests the getMemberById method by adding testMember and then retrieving it by its ID ("M1"). It ensures that the retrieved member is not null and that the memberId matches the expected value.

3.5 testGetAllMembers()

Tests the getAllMembers method by adding testMember and then retrieving all members from the database. It ensures that the list of members is not null, is not empty, and includes the test member.

3.6 testGetMembersByUnitId()

Tests the getMembersByUnitId method by adding testMember and retrieving members associated with unit "U1". It verifies that the list is not null, is not empty, and that the retrieved members have the correct unitId.

TESTING IMPLEMENTATION

4. NewsDAOTest.java

```java
@Test
public void testAddNews() {
    // Menggunakan berita yang sama
    Tblnews retrievedNews = newsDAO.getNewsById(testNews.getNewsId());
    assertNotNull(retrievedNews); // Pastikan berita tidak null
    assertEquals("Berita terbaru NCT", retrievedNews.getNews()); // Pastikan nama berita sesuai
}
```

```java
@Test
public void testUpdateNews() {
    testNews.setNews("Berita yang diperbarui");
    newsDAO.updateNews(testNews);
    Tblnews updatedNews = newsDAO.getNewsById(testNews.getNewsId());
    assertEquals("Berita yang diperbarui", updatedNews.getNews()); // Pastikan berita diperbarui
}
```

```java
@Test
public void testDeleteNews() {
    newsDAO.deleteNews(testNews);
    Tblnews deletedNews = newsDAO.getNewsById(testNews.getNewsId());
    assertNull(deletedNews); // Pastikan berita sudah dihapus
}
```

```java
@Test
public void testGetNewsById() {
    Tblnews news = newsDAO.getNewsById(testNews.getNewsId());
    assertNotNull(news); // Pastikan berita tidak null
    assertEquals("Berita terbaru NCT", news.getNews()); // Pastikan nama berita sesuai
}
```

```java
@Test
public void testGetAllNews() {
    // Mengambil semua berita dari database
    List<Tblnews> newsList = newsDAO.getAllNews();

    // Memastikan daftar berita tidak null
    assertNotNull(newsList);

    // Memastikan ada berita dalam daftar
    assertFalse(newsList.isEmpty());

    // Memastikan berita yang ditest ada dalam daftar
    boolean newsExists = newsList.stream().anyMatch(news -> news.getNewsId().equals(testNews.getNewsId()));
    assertTrue(newsExists); // Pastikan berita yang ditest ada dalam daftar
}
```

```java
@Test
public void testGetNewsByUnitId() {
    List<Tblnews> newsList = newsDAO.getNewsByUnitId("U1");
    assertNotNull(newsList); // Pastikan daftar berita tidak null
    assertFalse(newsList.isEmpty()); // Pastikan ada berita dalam daftar
    assertEquals("U1", newsList.get(0).getTblunit().getUnitId()); // Pastikan unit ID sesuai
}
```

TESTING IMPLEMENTATION

4.1 testAddNews()

Tests the addNews method by adding testNews to the database. It then retrieves the news by its ID and verifies that the news exists (assertNotNull) and the content matches the expected value ("Berita terbaru NCT").

4.2 testUpdateNews()

Tests the updateNews method by updating the content of testNews to "Berita yang diperbarui". It then retrieves the updated news and verifies that the content has been correctly updated.

4.3 testDeleteNews()

Tests the deleteNews method by deleting testNews. It verifies that the news is no longer retrievable from the database (it returns null).

4.4 testGetNewsById()

Tests the getNewsById method by retrieving testNews by its ID. It verifies that the news exists and that its content matches the expected value ("Berita terbaru NCT").

4.5 testGetAllNews()

Tests the getAllNews method by retrieving all news entries from the database. It ensures that the list of news is not null, is not empty, and that the testNews is included in the retrieved list.

4.6 testGetNewsByUnitId()

Tests the getNewsByUnitId method by retrieving news entries associated with unit "U1". It verifies that the list is not null, is not empty, and that the news entries are associated with the correct unit ID ("U1").

TESTING IMPLEMENTATION

5. UnitDAOTest.java

```
@Test
public void testGetAllUnits() {
    List<Tblunit> units = unitDAO.getAllUnits();
    assertNotNull(units);
    assertFalse(units.isEmpty());
    assertEquals(4, units.size()); // Karena ada 4 unit dalam gambar
}
```

```
@Test
public void testGetUnitsByUnitId() {
    List<Tblunit> nct127Units = unitDAO.getUnitsByUnitId("U1");
    assertNotNull(nct127Units);
    assertFalse(nct127Units.isEmpty());
    assertEquals(1, nct127Units.size());
    assertEquals("NCT 127", nct127Units.get(0).getUnitName());

    List<Tblunit> nctDreamUnits = unitDAO.getUnitsByUnitId("U2");
    assertNotNull(nctDreamUnits);
    assertFalse(nctDreamUnits.isEmpty());
    assertEquals(1, nctDreamUnits.size());
    assertEquals("NCT Dream", nctDreamUnits.get(0).getUnitName());

    // Disini saya juga bisa menambahkan pengujian untuk U3 dan U4 jika diperlukan
}
```

5.1 testGetAllUnits

This method tests the getAllUnits method of UnitDAO. Retrieves all units from the database using getAllUnits. The test asserts that the list of units is not null, the list is not empty (there are units in the database), the size of the list is 4, as the test expects there to be 4 units.

5.2 testGetUnitsByUnitId

Tests the getUnitsByUnitId method by retrieving units based on their unit ID. For unit ID "U1", it retrieves the unit and ensures the result is not null, the list is not empty, the list contains one unit, and that the unit name is "NCT 127". For unit ID "U2", it performs similar checks and ensures the unit name is "NCT Dream".

TESTING IMPLEMENTATION

6. TbladminTest.java

```
@Test
public void testValidasiLogin() {
    System.out.println("validasiLogin");

    // Pengujian untuk login yang berhasil
    instance = new Tbladmin("faza_and_via", "fv@nctadmin.com", "123", "admin");
    String expResult = "admin?faces-redirect=true";
    String result = instance.validasiLogin();
    assertEquals(expResult, result);


}

@Test
public void testValidasiLoginWithInvalidCredentials() {
    instance.setUsername("invalid_user");
    instance.setPassword("wrong_password");
    String expResult = "signin.xhtml?faces-redirect=true";
    String result = instance.validasiLogin();
    assertEquals(expResult, result);
}
```

6.1 testValidasiLogin()

This method tests the login functionality with valid credentials. A new instance of Tbladmin is created with the username "faza_and_via", email "fv@nctadmin.com", password "123", and role "admin". The expected result is "admin?faces-redirect=true", which is typically the navigation outcome for a successful login (redirecting to an admin page). The actual result is obtained by calling the validasiLogin() method from the Tbladmin instance. The assertEquals() method checks whether the expected result matches the actual result.

6.2 testValidasiLoginWithInvalidCredentials()

This method tests the login functionality with invalid credentials. The username is set to "invalid_user" and the password to "wrong_password", simulating a user trying to log in with incorrect credentials. The expected result is "signin.xhtml?faces-redirect=true", which indicates the user should be redirected to the sign-in page upon a failed login attempt. The actual result is again obtained by calling validasiLogin(). The assertEquals() method checks whether the expected result matches the actual result.

TESTING IMPLEMENTATION

```
@Test
public void testLogout() {
    System.out.println("logout");
    String expResult = "/index.xhtml?faces-redirect=true";
    String result = instance.logout();
    assertEquals(expResult, result);
}

/**
 * Test of isLoggedIn method, of class Tbladmin.
 */
@Test
public void testIsLoggedIn() {
    boolean expResult = true;
    boolean result = instance.isLoggedIn();
    assertEquals(expResult, result);
}

/**
 * Test of afterPhase method, of class Tbladmin.
 */
@Test
public void testAfterPhase() {
    // Buat mock FacesContext
    FacesContext mockContext = Mockito.mock(FacesContext.class);
    UIViewRoot mockViewRoot = Mockito.mock(UIViewRoot.class);
    // Set up mock behavior
    Mockito.when(mockContext.getViewRoot()).thenReturn(mockViewRoot);
    Mockito.when(mockViewRoot.getViewId()).thenReturn("/admin.xhtml");
    // Buat PhaseEvent dengan mock FacesContext
    PhaseEvent event = new PhaseEvent(mockContext, PhaseId.RESTORE_VIEW, Mockito.mock(Lifecycle.class));
    // Jalankan metode afterPhase
    instance.afterPhase(event);
    // Tidak ada assertion yang diperlukan jika tidak ada pengecualian yang dilempar
}
```

6.3 testLogout()

This method tests the logout() functionality of the Tbladmin class. It expects the logout method to return a redirect to the index page ("/index.xhtml?faces-redirect=true"). The actual result is obtained by calling the logout() method from the Tbladmin instance. The assertEquals() method checks if the actual result matches the expected redirect outcome.

6.4 testIsLoggedIn()

This method tests the isLoggedIn() method of the Tbladmin class. It expects the isLoggedIn() method to return true, indicating that the user is logged in. The actual result is obtained by calling isLoggedIn() from the Tbladmin instance. The assertEquals() method checks whether the expected result (true) matches the actual result.

TESTING IMPLEMENTATION

6.5 testAfterPhase()

Purpose: This method tests the afterPhase() method, which is likely part of the JSF lifecycle handling. Mocks are created for FacesContext, UIViewRoot, and Lifecycle using Mockito. The mock behavior is set up to return a view ID ("/admin.xhtml") when getViewId() is called on the mocked UIViewRoot. A PhaseEvent is created using the mock FacesContext and Lifecycle, passing PhaseId.RESTORE_VIEW (indicating the phase in the JSF lifecycle). The afterPhase() method of the Tbladmin instance is invoked with the event, simulating behavior during a specific phase of the JSF lifecycle. No assertion is needed as the test passes if no exceptions are thrown during the method execution.

TESTING IMPLEMENTATION

7. TblalbumTest.java

```
@Test
public void testGetNct127Albums() {
    List<Tblalbum> mockAlbums = new ArrayList<>();
    when(albumDAO.getAlbumsByUnitId("u1")).thenReturn(mockAlbums);
    List<Tblalbum> result = tblalbum.getNct127Albums();
    assertSame(mockAlbums, result);
    verify(albumDAO).getAlbumsByUnitId("u1");
}

@Test
public void testGetNctDreamAlbums() {
    List<Tblalbum> mockAlbums = new ArrayList<>();
    when(albumDAO.getAlbumsByUnitId("u2")).thenReturn(mockAlbums);
    List<Tblalbum> result = tblalbum.getNctDreamAlbums();
    assertSame(mockAlbums, result);
    verify(albumDAO).getAlbumsByUnitId("u2");
}

@Test
public void testGetNctUAlbums() {
    List<Tblalbum> mockAlbums = new ArrayList<>();
    when(albumDAO.getAlbumsByUnitId("u3")).thenReturn(mockAlbums);
    List<Tblalbum> result = tblalbum.getNctUAlbums();
    assertSame(mockAlbums, result);
    verify(albumDAO).getAlbumsByUnitId("u3");
}

@Test
public void testGetWayVAlbums() {
    List<Tblalbum> mockAlbums = new ArrayList<>();
    when(albumDAO.getAlbumsByUnitId("u4")).thenReturn(mockAlbums);
    List<Tblalbum> result = tblalbum.getWayVAlbums();
    assertSame(mockAlbums, result);
    verify(albumDAO).getAlbumsByUnitId("u4");
}
```

TESTING IMPLEMENTATION

7.1 testGetNct127Albums()

This method tests the functionality for retrieving albums by the NCT 127 unit. A mock list of Tblalbum objects is created (mockAlbums). The albumDAO.getAlbumsByUnitId("u1") is stubbed using when(...).thenReturn(...) to return this mock list when called with the unit ID "U1". The tblalbum.getNct127Albums() method is called to get the result, and the test checks if this result is the same object as mockAlbums using assertSame(). Also, verify(albumDAO) ensures that the getAlbumsByUnitId("U1") method is called once during the process.

7.2 testGetNctDreamAlbums()

This method tests the functionality for retrieving albums by the NCT Dream unit. Similar to the first method, a mock list of albums (mockAlbums) is created. The albumDAO.getAlbumsByUnitId("u2") method is stubbed to return this list for the NCT Dream unit with ID "U2". The tblalbum.getNctDreamAlbums() method is invoked, and assertSame() ensures the result is the mock list. Als, verify(albumDAO) checks that getAlbumsByUnitId("U2") was invoked.

7.3 testGetNctUAlbums()

This method tests the functionality for retrieving albums by the NCT U unit. A mock list of albums (mockAlbums) is set up.The albumDAO.getAlbumsByUnitId("U3") method is stubbed to return the list for the NCT U unit with ID "U3".The tblalbum.getNctUAlbums() method is called, and assertSame() checks that the returned result is the same mock list. Also, verify(albumDAO) checks that getAlbumsByUnitId("U3") was called.

7.4 testGetWayVAlbums()

This method tests the functionality for retrieving albums by the WayV unit. A mock list of albums (mockAlbums) is created.The albumDAO.getAlbumsByUnitId("U4") method is stubbed to return this list for the WayV unit with ID "U4".The tblalbum.getWayVAlbums() method is invoked, and assertSame() ensures the result matches the mock list.

TESTING IMPLEMENTATION

```
@Test
public void testAddNewAlbum() throws IOException {
    Tblalbum newAlbum = new Tblalbum();
    newAlbum.setAlbumName("New Test Album");
    newAlbum.setTblunit(new Tblunit());
    newAlbum.getTblunit().setUnitId("U1");

    when(albumImageFile.getInputStream()).thenReturn(new ByteArrayInputStream(new byte[]{1, 2, 3, 4, 5}));
    when(albumImageFile.getSize()).thenReturn(5L);

    tblalbum.setNewAlbum(newAlbum);
    tblalbum.setAlbumImageFile(albumImageFile);
    tblalbum.addNewAlbum();

    verify(albumDAO).addAlbum(newAlbum);
    assertNotNull(newAlbum.getAlbumImg());
    assertEquals(5, newAlbum.getAlbumImg().length);

    // Pengujian untuk kasus error
    doThrow(new RuntimeException("Test Exception")).when(albumDAO).addAlbum(any(Tblalbum.class));
    tblalbum.addNewAlbum();
    verify(facesContextWrapper, times(2)).addMessage(eq(null), any(FacesMessage.class));
}
```

```
@Test
public void testUpdateAlbum() {
    Tblalbum album = new Tblalbum();
    album.setAlbumId(1);
    album.setAlbumName("Test Album");

    tblalbum.updateAlbum(album);

    verify(albumDAO).updateAlbum(album);
    verify(facesContextWrapper).addMessage(eq(null), argThat(message ->
        message.getSeverity() == FacesMessage.SEVERITY_INFO &&
        "Success".equals(message.getSummary()) &&
        "Album berhasil diperbarui".equals(message.getDetail())
    ));

    // Pengujian untuk kasus error
    doThrow(new RuntimeException("Test Exception")).when(albumDAO).updateAlbum(any(Tblalbum.class));
    tblalbum.updateAlbum(album);
    verify(facesContextWrapper, times(2)).addMessage(eq(null), any(FacesMessage.class));
}
```

TESTING IMPLEMENTATION

```
@Test
public void testDeleteAlbum() {
    Tblalbum album = new Tblalbum();
    album.setAlbumId(1);

    tblalbum.albums = new ArrayList<>(Arrays.asList(album));
    tblalbum.deleteAlbum(album);

    verify(albumDAO).deleteAlbum(album);
    assertTrue(tblalbum.albums.isEmpty());
    verify(facesContextWrapper).addMessage(eq(null), any(FacesMessage.class));

    // Pengujian untuk kasus error
    doThrow(new RuntimeException("Test Exception")).when(albumDAO).deleteAlbum(any(Tblalbum.class));
    tblalbum.deleteAlbum(album);
    verify(facesContextWrapper, times(2)).addMessage(eq(null), any(FacesMessage.class));
}
```

```
@Test
public void testHandleFileUpload() throws IOException {
    Tblalbum album = new Tblalbum();
    album.setAlbumId(1);

    when(albumImageFile.getInputStream()).thenReturn(new ByteArrayInputStream(new byte[]{1, 2, 3, 4, 5}));
    when(albumImageFile.getSize()).thenReturn(5L);

    tblalbum.setAlbumImageFile(albumImageFile);
    tblalbum.handleFileUpload(album);

    verify(albumDAO).updateAlbum(album);
    assertNotNull(album.getAlbumImg());
    assertEquals(5, album.getAlbumImg().length);
}
```

7.5 testAddNewAlbum()

Tests the addition of a new album and handles potential errors. Prepares a Tblalbum object with a mock image file. Mocks file input streams and verifies that the album image is correctly added and has a valid byte size. Verifies that the addAlbum() method is called and displays appropriate success or error messages.

7.6 testUpdateAlbum()

Tests the update functionality for an existing album and handles exceptions. Prepares a Tblalbum object to be updated. Calls the updateAlbum() method and verifies the update process and success message. Tests error handling by throwing a mock exception and checking that the error message is displayed.

TESTING IMPLEMENTATION

7.7 testDeleteAlbum()

Tests the deletion of an album and handles errors. Prepares a Tblalbum object for deletion. Verifies that the deleteAlbum() method is called and the album is removed from the list. Simulates an error during deletion and checks that the error message is displayed.

7.8 testHandleFileUpload()

Tests the handleFileUpload method. It ensures the image file is uploaded and the album is updated with the image data.

TESTING IMPLEMENTATION

8. TblmemberTest.java

```
@Test
public void testAddNewMember() {
    // Persiapan
    Tblmember newMember = new Tblmember();
    newMember.setMemberName("New Member");
    newMember.setRole("New Role");
    Tblunit unit = new Tblunit();
    unit.setUnitId("U1");
    newMember.setTblunit(unit);

    member.setNewMember(newMember);

    when(memberDAO.getAllMembers()).thenReturn(Arrays.asList(
        new Tblmember("M1", "Existing Member", "Role")
    ));
    doNothing().when(memberDAO).addMember(any(Tblmember.class));

    // Eksekusi
    member.addNewMember();

    // Verifikasi
    verify(memberDAO).addMember(any(Tblmember.class));
    verify(memberDAO, times(2)).getAllMembers(); // Dipanggil untuk generateNextMemberId dan refresh
    verify(facesContextWrapper).addMessage(eq(null), any(FacesMessage.class));

    // Pengujian untuk kasus error
    doThrow(new RuntimeException("Test Exception")).when(memberDAO).addMember(any(Tblmember.class));
    member.addNewMember();
    verify(facesContextWrapper, times(2)).addMessage(eq(null), any(FacesMessage.class)); // Verifikasi pesan error ditambahkan
}
```

```
@Test
public void testUpdateMember() {
    // Persiapan
    Tblmember memberToUpdate = new Tblmember("M1", "Updated Member", "Updated Role");
    doNothing().when(memberDAO).updateMember(any(Tblmember.class));
    when(memberDAO.getAllMembers()).thenReturn(Arrays.asList(memberToUpdate));

    // Eksekusi
    member.updateMember(memberToUpdate);

    // Verifikasi
    verify(memberDAO).updateMember(memberToUpdate);
    verify(memberDAO).getAllMembers();
    verify(facesContextWrapper).addMessage(eq(null), any(FacesMessage.class));

    // Pengujian untuk kasus error
    doThrow(new RuntimeException("Test Exception")).when(memberDAO).updateMember(any(Tblmember.class));
    member.updateMember(memberToUpdate);
    verify(facesContextWrapper, times(2)).addMessage(eq(null), any(FacesMessage.class)); // Verifikasi pesan error ditambahkan
}
```

TESTING IMPLEMENTATION

```
@Test
public void testDeleteMember() {
    // Persiapan
    Tblmember memberToDelete = new Tblmember("M1", "Member to Delete", "Role");
    doNothing().when(memberDAO).deleteMember(any(Tblmember.class));
    member.members = new ArrayList<>(Arrays.asList(memberToDelete));

    // Eksekusi
    member.deleteMember(memberToDelete);

    // Verifikasi
    verify(memberDAO).deleteMember(memberToDelete);
    verify(facesContextWrapper).addMessage(eq(null), any(FacesMessage.class));
    assertTrue(member.members.isEmpty());

    // Pengujian untuk kasus error
    doThrow(new RuntimeException("Test Exception")).when(memberDAO).deleteMember(any(Tblmember.class));
    member.deleteMember(memberToDelete);
    verify(facesContextWrapper, times(2)).addMessage(eq(null), any(FacesMessage.class)); // Verifikasi pesan error ditambahkan
}
```

8.1 testAddNewMember()

Tests the addNewMember() method for both successful and exception cases when adding a new member. A new member (Tblmember) is created and assigned a unit. Mocks are set up to simulate existing members and a successful addition using memberDAO. Verifies that the memberDAO.addMember() method is called once to add the new member, and the list of members is retrieved twice (for ID generation and refreshing). It also checks that a success message is displayed after the member is added. In the second part, an exception is simulated during the member addition, and the test verifies that an error message is shown.

8.2 testUpdateMember()

Tests the updateMember() method for both successful and exception cases. Prepares a Tblmember object to be updated. Mocks the updateMember() method to simulate successful updating. Verifies that the member is updated and the list of members is refreshed. Also simulates an error case where an exception is thrown, verifying that an error message is displayed.

TESTING IMPLEMENTATION

8.3 testDeleteMember()

Tests the deleteMember() method for both successful and exception cases. Prepares a Tblmember object for deletion. Mocks the deleteMember() method to simulate a successful deletion. Verifies that the member is deleted from the list and a success message is shown. For the error case, it simulates an exception during deletion and checks that an error message is displayed.

TESTING IMPLEMENTATION

9. TblnewsTest.java

```
@Test
public void testUpdateNews() {
    Tblnews news = new Tblnews();
    news.setNewsId(1);
    news.setNews("Test News");
    List<Tblnews> mockList = new ArrayList<>();
    mockList.add(news);
    tblnews.setNewsList(mockList);

    tblnews.updateNews(news);

    verify(newsDAO).updateNews(news);
    verify(facesContextWrapper).addMessage(eq(null), argThat(message ->
        message.getSeverity() == FacesMessage.SEVERITY_INFO &&
        "Success".equals(message.getSummary()) &&
        "Berita berhasil diperbarui".equals(message.getDetail())
    ));
    assertEquals(news, mockList.get(0));
}
```

```
@Test
public void testUpdateNewsException() {
    Tblnews news = new Tblnews();
    doThrow(new RuntimeException("Update failed")).when(newsDAO).updateNews(news);

    tblnews.updateNews(news);

    verify(facesContextWrapper).addMessage(eq(null), argThat(message ->
        message.getSeverity() == FacesMessage.SEVERITY_ERROR &&
        "Error".equals(message.getSummary()) &&
        "Gagal memperbarui berita: Update failed".equals(message.getDetail())
    ));
}
```

TESTING IMPLEMENTATION

```
@Test
public void testDeleteNews() {
    Tblnews news = new Tblnews();
    news.setNewsId(1);
    List<Tblnews> mockList = new ArrayList<>();
    mockList.add(news);
    tblnews.setNewsList(mockList);

    tblnews.deleteNews(news);

    verify(newsDAO).deleteNews(news);
    verify(facesContextWrapper).addMessage(eq(null), argThat(message ->
        message.getSeverity() == FacesMessage.SEVERITY_INFO &&
        "Success".equals(message.getSummary()) &&
        "Berita berhasil dihapus".equals(message.getDetail())
    ));
    assertTrue(mockList.isEmpty());
}
```

```
@Test
public void testDeleteNewsException() {
    Tblnews news = new Tblnews();
    doThrow(new RuntimeException("Delete failed")).when(newsDAO).deleteNews(news);

    tblnews.deleteNews(news);

    verify(facesContextWrapper).addMessage(eq(null), argThat(message ->
        message.getSeverity() == FacesMessage.SEVERITY_ERROR &&
        "Error".equals(message.getSummary()) &&
        "Gagal menghapus berita: Delete failed".equals(message.getDetail())
    ));
}
```

TESTING IMPLEMENTATION

9.1 testUpdateNews()

Tests the updateNews() method for successful news updates. Mocks a news update, verifies that the newsDAO.updateNews() method is called, and checks if a success message ("Berita berhasil diperbarui") is added. Ensures the news list is correctly updated.

9.2 testUpdateNewsException()

Tests the updateNews() method when an exception occurs during the update. Simulates an exception thrown by newsDAO.updateNews() and verifies that an error message ("Gagal memperbarui berita") is shown.

9. 3 testDeleteNews()

Tests the deleteNews() method for successful deletion of news.es: Mocks the news deletion, verifies that newsDAO.deleteNews() is called, ensures a success message ("Berita berhasil dihapus") is displayed, and confirms that the news list is empty after deletion.

9.4 testDeleteNewsException()

Tests the deleteNews() method when an exception occurs during deletion. Simulates an exception thrown by newsDAO.deleteNews() and verifies that an error message ("Gagal menghapus berita") is shown.

TESTING IMPLEMENTATION

10. TblunitTest.java

```
@Test
public void testGetUnits() {
    List<Tblunit> units = tblunit.getUnits();
    assertNotNull(units);
    assertFalse(units.isEmpty());
}
@Test
public void testGetNct127Units() {
    List<Tblunit> units = tblunit.getNct127Units();
    assertNotNull(units);
    assertEquals(1, units.size());
    assertEquals("NCT 127", units.get(0).getUnitName());
}
@Test
public void testGetDreamUnits() {
    List<Tblunit> units = tblunit.getDreamUnits();
    assertNotNull(units);
    assertEquals(1, units.size());
    assertEquals("NCT Dream", units.get(0).getUnitName());
}
@Test
public void testGetNctUnits() {
    List<Tblunit> units = tblunit.getNctUnits();
    assertNotNull(units);
    assertEquals(1, units.size());
    assertEquals("NCT U", units.get(0).getUnitName());
}
@Test
public void testGetWayVUnits() {
    List<Tblunit> units = tblunit.getWayVUnits();
    assertNotNull(units);
    assertEquals(1, units.size());
    assertEquals("WAYV", units.get(0).getUnitName());
}
```

TESTING IMPLEMENTATION

10.1 testGetUnits()

Tests that the getUnits() method returns a non-null and non-empty list of all units. Ensures that the getUnits() method provides valid unit data.

10.2 testGetNct127Units()

Verifies that the getNct127Units() method returns the NCT 127 unit. What it does: Asserts that the method returns a list containing exactly one unit with the name "NCT 127".

10.3 testGetDreamUnits()

Confirms that the getDreamUnits() method returns the NCT Dream unit. Asserts that the list contains exactly one unit with the name "NCT Dream".

10.4 testGetNctUnits()

Tests that the getNctUnits() method retrieves the NCT U unit. Verifies the list contains exactly one unit named "NCT U".

10.5 testGetWayVUnits()

Ensures that the getWayVUnits() method retrieves the WayV unit. Asserts that the method returns a list with one unit named "WAYV".

TESTING IMPLEMENTATION

index.xhtml

No.	Test Case	Test Data	Expected Result	Actual Resut	Result
1	user Sign in by clicking Sign In button	User click Sign In button	go to signin.xhtml page	go to signin.xhtml page	PASS

signin.xhtml

No.	Test Case	Test Data	Expected Result	Actual Result	Result
1	user input username and password into the Sign In form and click the Sign In button	User fill the valid username and password	the system will go to the admin page and display the username on the top section of the page	the system will go to the admin page and display the username on the top section of the page	PASS
2	user input username and password into the Sign In form and click the submit button	User fill the wrong username and password	the system will show "Login Failed"	the system will show "Login Failed"	PASS
3	User click the cancel button in the Sign In page	Use click the cancel button	the system will display the index page	the system will display the index page	PASS

TESTING IMPLEMENTATION

admin.xhtml

No.	Test Case	Test Data	Expected Result	Actual Result	Result
1	user add a new album by filling the content	User input the content data: album image, album title, description, album links, and unit ID	stay in admin page and the album data will be updated, and the inputed data will be saved in the database	stay in admin page and the unit data will be updated, and the inputed data will be saved in the database	PASS
2	user add a new album with empty content data	The content left empty	stay in admin page and the album data will be updated, and the data will be stay unchanged	stay in admin page and the album data will be updated, and the inputed empty data loaded	FAILED
3	user edit the data of an album and click save button	User change the album data	stay in admin page and the album data will be updated, and the edited data will be saved in the database	stay in admin page and the unit data will be updated, and the edited data will be saved in the database	PASS
4	user delete an album by clicking the delete button	user delete an album	stay in admin page and the album data will be updated, and the chosen data will be deleted in the database	stay in admin page and the unit data will be updated, and the chosen data will be deleted in the database	PASS

TESTING IMPLEMENTATION

admin.xhtml

No.	Test Case	Test Data	Expected Result	Actual Result	Result
5	user add a new news row by filling news data and click the Add News button	User input the content data: news image, news content, Unit ID	stay in admin page and the news data will be updated, and the inputed data will be saved in the database. The inputed data will be showed in the News page	stay in admin page and the news data will be updated, and the inputed data will be saved in the database. The inputed data does not shown in the news page	PASS
6	user add a new news row by left the news data empty and click the Add News button	The content left empty	stay in admin page and the news data will be updated, and the data will be stay unchanged	stay in admin page and the news data will be updated, and the inputed empty data loaded	FAILED
7	user edit either the news image or news content and then click the save button	User change a news image/news content	stay in admin page and the news data will be updated, and the edited data will be saved in the database. The data will be updated in the News page	stay in admin page and the news data will be updated, and the edited data will be saved in the database. The data will be updated in the News page	PASS

TESTING IMPLEMENTATION

admin.xhtml

No.	Test Case	Test Data	Expected Result	Actual Resut	Result
8	user delete a news by clicking the delete button	user delete a news	stay in admin page and the news data will be updated, the chosen data will be deleted in the database and in the News page.	stay in admin page and the news data will be updated, the chosen data will be deleted in the database and in the News page.	PASS
9	user add a new member by inputing the data and click add new member button	user input the content data: member name, image address, role, and unit	stay in admin page with "Success" message and the data will be updated both in the page and in the database. The data will be shown in the each Unit's page	stay in admin page with "Success" message and the data will be updated both in the page and in the database. The data will be shown in the each Unit's page	PASS
10	user add a new member by left the data empty and click add new member button	The content left empty	stay in admin page and the news data will be updated, and the data will be stay unchanged	stay in admin page with "Success" message and the data will be updated both in the page and in the database with null data. The data will be shown in the each Unit's page	FAILED

TESTING IMPLEMENTATION

admin.xhtml

No.	Test Case	Test Data	Expected Result	Actual Resut	Result
11	user edit a member's name, image address, role and click the save button	user change a member's name, image address, or role, or all of it	stay in admin page with "Success" message and the data will be updated both in the admin page and in the database. The data also will be updated in the each Unit's page	stay in admin page with "Success" message and the data will be updated both in the admin page and in the database. The data also will be updated in the each Unit's page	PASS
12	user delete a member by clicking delete button	user delete a member	stay in admin page with "Success" message and the data will be updated both in the admin page and in the database. The data also will be deleted in the each Unit's page	stay in admin page with "Success" message and the data will be updated both in the admin page and in the database. The data also will be deleted in the each Unit's page	PASS
13	user logging out by clicking Log Out button in the admin.xhtml page	user click Log Out button	go to index.xhtml page	go to index.xhtml page	PASS

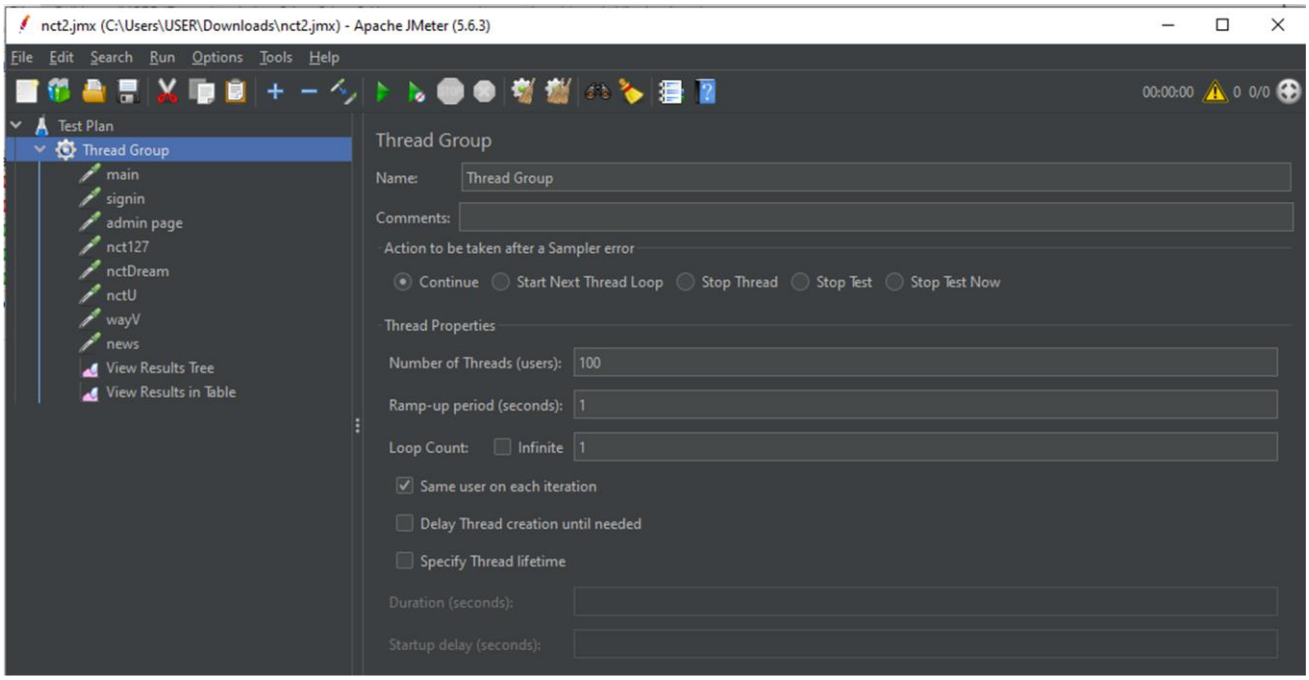
TESTING IMPLEMENTATION

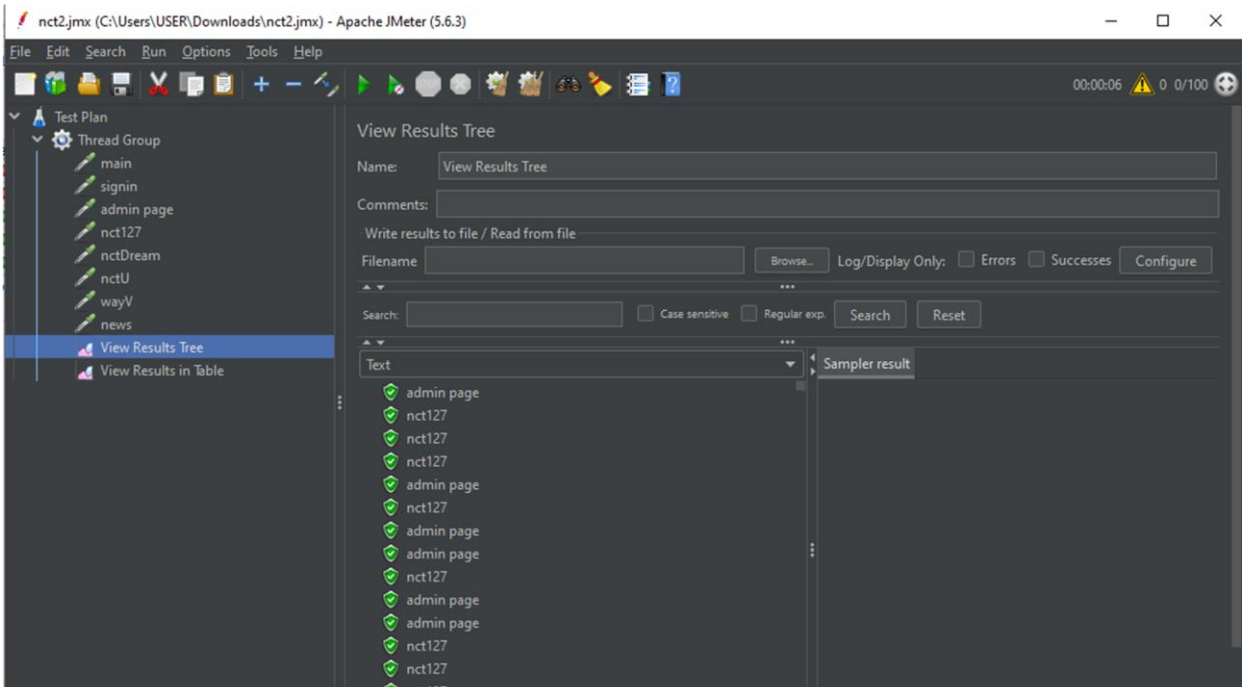
header.xhtml

No.	Test Case	Test Data	Expected Result	Actual Result	Result
1	go to news.xhtml page by click News text on the navigation bar	click News text on the navigation bar	display news.xhtml page	display news.xhtml page	PASS
2	go to nct127.xhtml page by click NCT 127 text on the navigation bar	click NCT 127 text on the navigation bar	display nct127.xhtml page	display nctDream.xhtml page	PASS
3	go to nctDream.xhtml page by click NCT Dream text on the navigation bar	click NCT Dream text on the navigation bar	display nctDream.xhtml page	display nct127.xhtml page	PASS
4	go to nctU.xhtml page by click NCT U text on the navigation bar	click NCT U text on the navigation bar	display nctU.xhtml page	display nctU.xhtml page	PASS
5	go to wayV.xhtml page by click WayV text on the navigation bar	click WayV text on the navigation bar	display wayV.xhtml page	display wayV.xhtml page	PASS
6	go to admin.xhtml page by click Admin text on the navigation bar	click Admin text on the navigation bar	display admin.xhtml page	display admin.xhtml page	PASS

TESTING IMPLEMENTATION

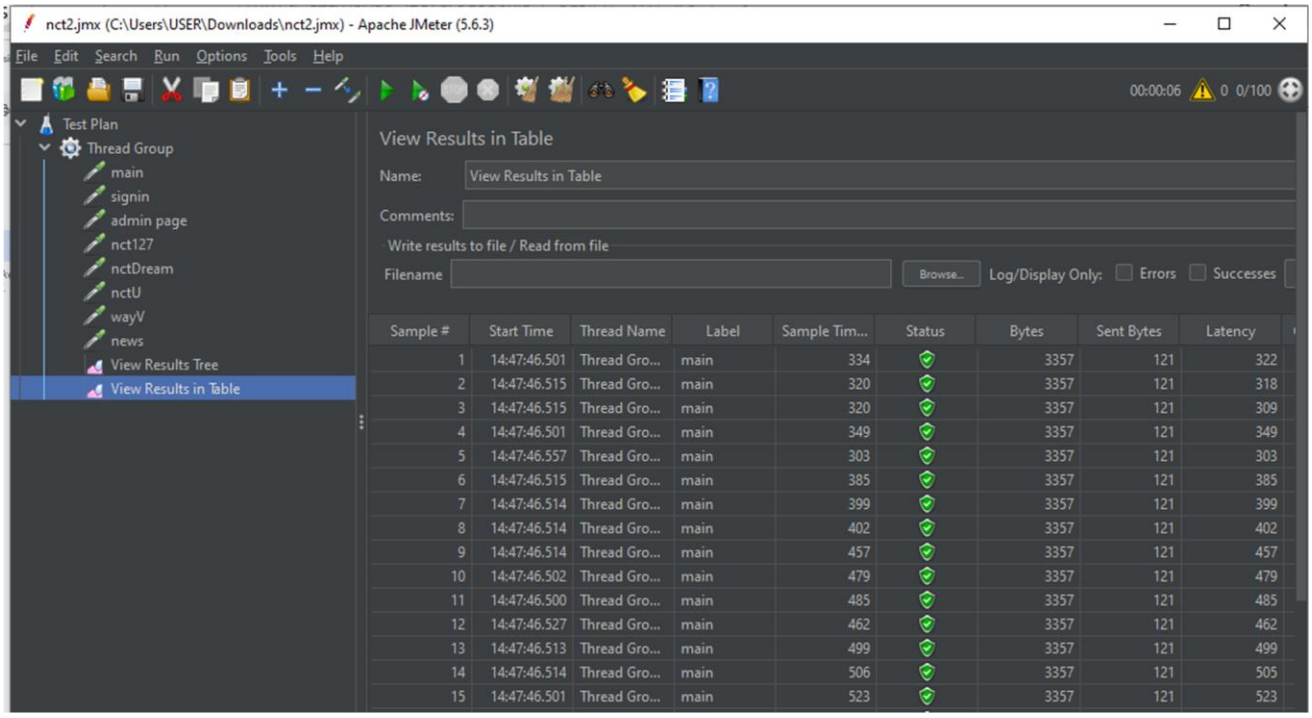
JMeter Testing





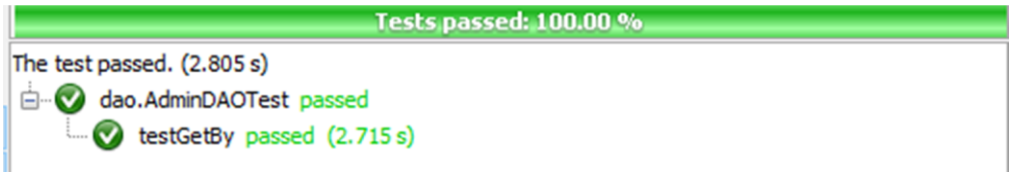
RESULT

JMeter Testing

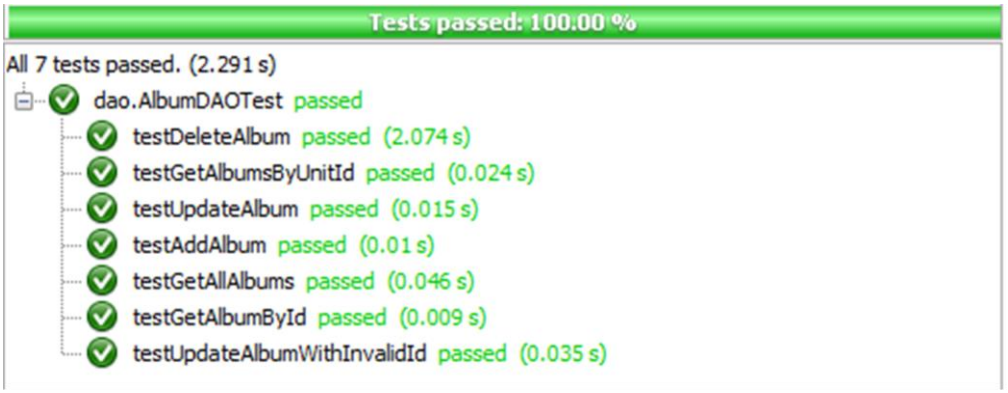


RESULT

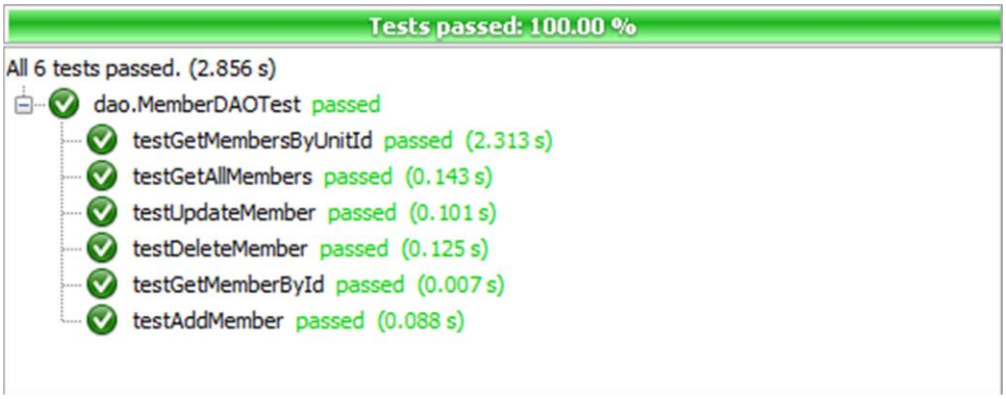
1. AdminDAOTest.java



2. AlbumDAOTest.java

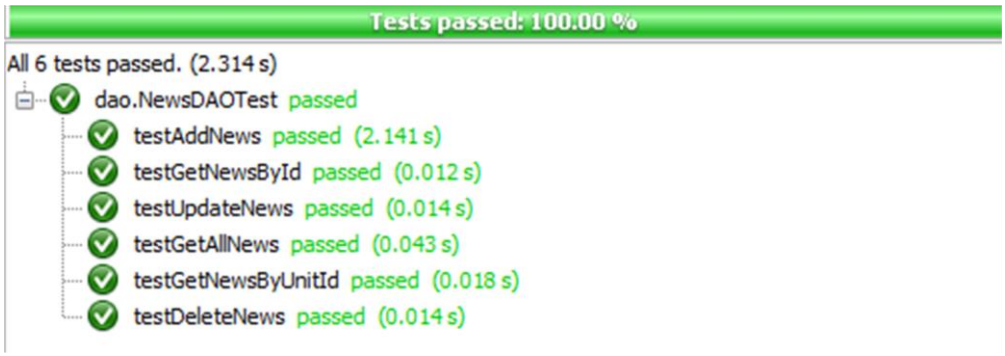


3. MemberDAOTest.java



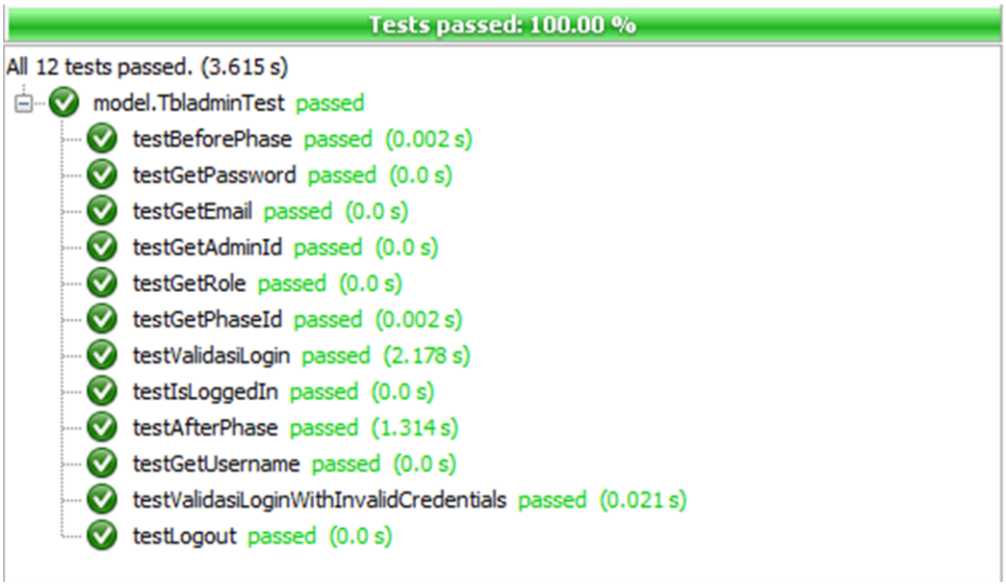
RESULT

4. NewsDAOTest.java



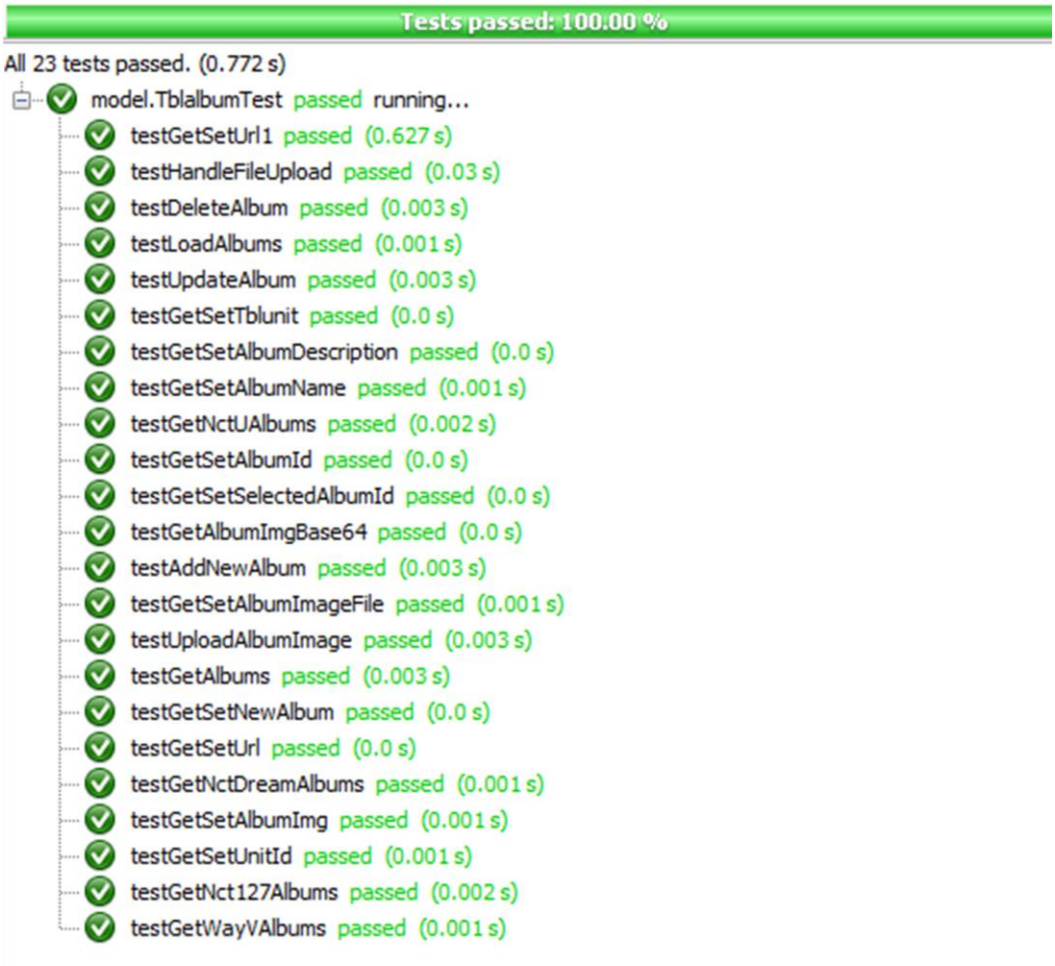
5. UnitDAOTest.java

6. TbladminTest.java



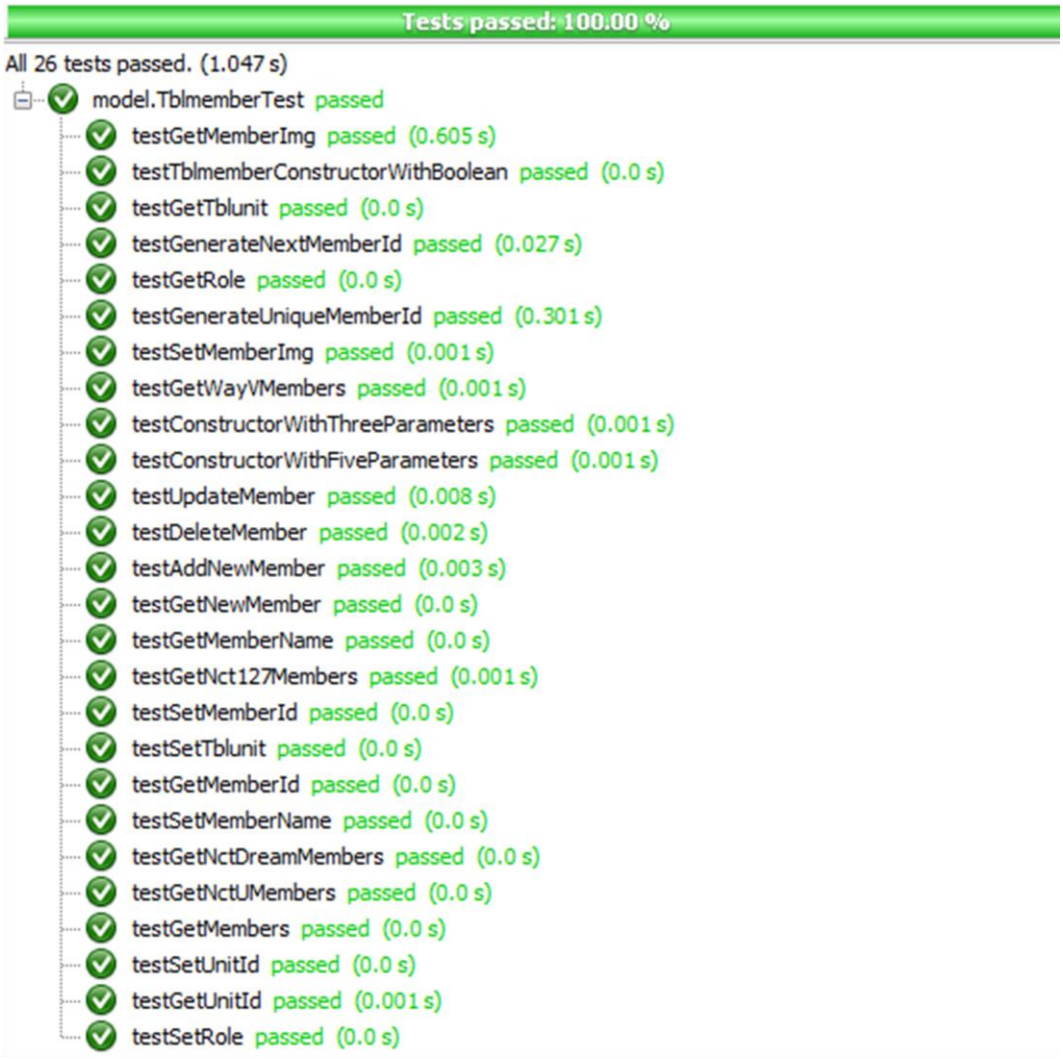
RESULT

7. TblalbumTest.java



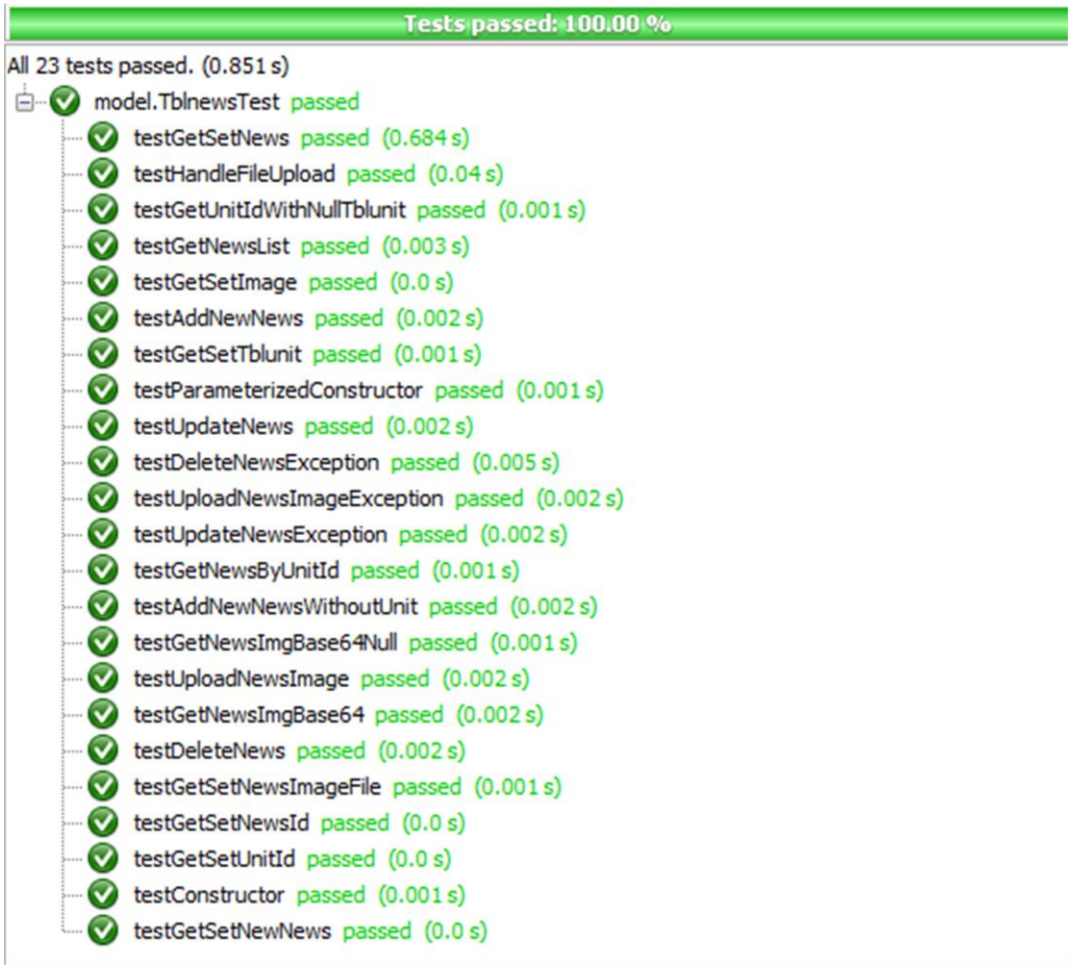
RESULT

8. TblmemberTest.java



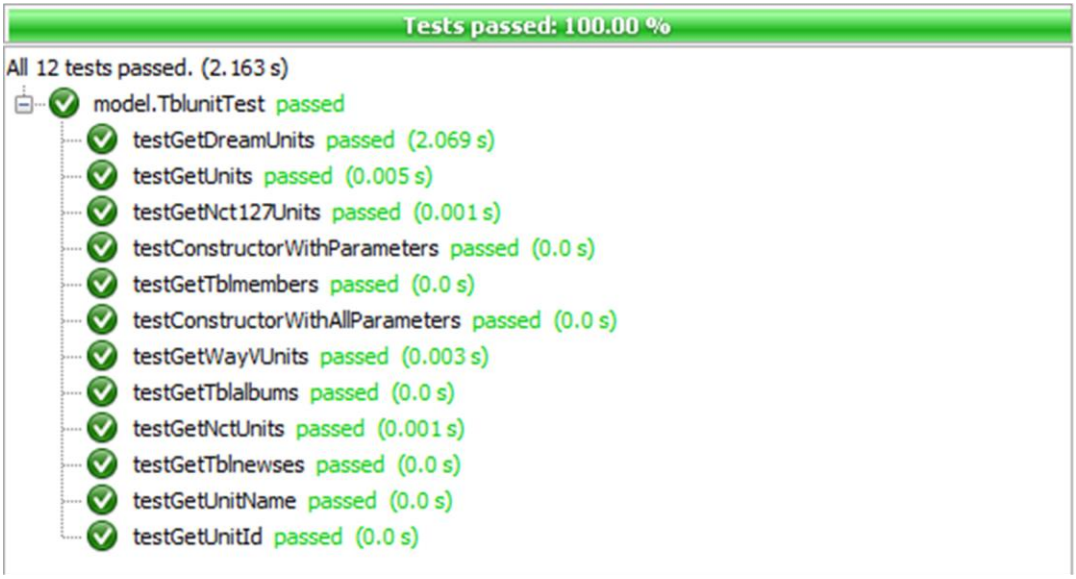
RESULT

9. TblnewsTest.java



RESULT

10. TblunitTest.java



RESULT JaCoCo

JaCoCoverage analysis of project "nct2" (powered by JaCoCo from EclEmma)

JaCoCoverage analysis of project "nct2" (powered by JaCoCo from EclEmma)

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
dao		59%		22%	102	134	54	249	0	31	0	5
model		82%		49%	61	199	32	431	1	129	0	5
util		57%		50%	5	12	10	22	4	10	0	3
Total	767 of 2,814	73%	233 of 350	33%	168	345	96	702	5	170	0	13

model

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Tblnews		70%		46%	22	53	10	111	0	27	0	1
Tblalbum		79%		44%	17	56	12	126	1	38	0	1
Tbladmin		83%		50%	16	35	6	58	0	18	0	1
Tblmember		95%		69%	5	36	4	97	0	28	0	1
Tblunit		100%		50%	1	19	0	39	0	18	0	1
Total	316 of 1,720	82%	71 of 140	49%	61	199	32	431	1	129	0	5

JaCoCoverage analysis of project "nct2" (powered by JaCoCo from EclEmma) > dao

dao

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
NewsDAO		55%		20%	33	41	17	69	0	8	0	1
MemberDAO		56%		21%	31	39	18	70	0	8	0	1
AlbumDAO		62%		24%	33	41	13	71	0	8	0	1
UnitDAO		60%		25%	4	9	6	21	0	5	0	1
AdminDAO		83%		75%	1	4	2	18	0	2	0	1
Total	413 of 1,020	60%	159 of 206	23%	102	134	56	249	0	31	0	5