```java
package au.edu.sydney.soft3202.reynholm.erp.billingsystem;

import au.edu.sydney.soft3202.reynholm.erp.client.ClientReporting;
import au.edu.sydney.soft3202.reynholm.erp.compliance.ComplianceReporting;
import au.edu.sydney.soft3202.reynholm.erp.permissions.AuthToken;
import au.edu.sydney.soft3202.reynholm.erp.permissions.AuthenticationModule;
import au.edu.sydney.soft3202.reynholm.erp.permissions.AuthorisationModule;
import au.edu.sydney.soft3202.reynholm.erp.project.Project;
import au.edu.sydney.soft3202.reynholm.erp.cheatmodule.ERPCheatFactory;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;
import org.mockito.MockedStatic;
import org.mockito.junit.jupiter.MockitoExtension;

import java.util.List;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;
import static org.mockito.Mockito.mockStatic;

@ExtendWith(MockitoExtension.class)
public class BSFacadeImplTest {
    private final ERPCheatFactory erp = new ERPCheatFactory();

    //Mocks
    private AuthenticationModule authenticationModule;
    private AuthorisationModule authorisationModule;
    private AuthToken basicAuthToken;
    private AuthToken secureAuthToken;
    private AuthToken basicAndSecure;
    private ClientReporting clientReporting;
    private ComplianceReporting complianceReporting;

    @InjectMocks
    private BSFacadeImpl bsf = new BSFacadeImpl();

    ////////////////////////////////////////////////////////////////////
    // Set mocks
    ////////////////////////////////////////////////////////////////////
    @BeforeEach
    public void setMocks(){
        //create mocks
        authenticationModule = mock(AuthenticationModule.class);
        authorisationModule = mock(AuthorisationModule.class);

        basicAuthToken = mock(AuthToken.class);
        secureAuthToken = mock(AuthToken.class);
        basicAndSecure = mock(AuthToken.class);

        clientReporting = mock(ClientReporting.class);
        complianceReporting = mock(ComplianceReporting.class);

        bsf = new BSFacadeImpl();
    }

    ////////////////////////////////////////////////////////////////////
```

```
    // Basic Tests
    ////////////////////////////////////////////////////////////////////////
    ////////////////////////////////////////////////////////////////////////
    // Add Project Tests
    ////////////////////////////////////////////////////////////////////////
    @Test
    public void addProjectNoPermsMod(){
        assertThrows(IllegalStateException.class, () ->
bsf.addProject("projectName", "clientName", 50.0d, 70.0d));
    }

    @Test
    public void addProjectNoLoggedInUser(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        assertThrows(IllegalStateException.class, () ->
bsf.addProject("projectName", "clientName", 50.0d, 70.0d));
    }

    @Test
    public void addProjectProjectNameNull(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        assertThrows(IllegalArgumentException.class, () -> bsf.addProject(null,
"clientName", 50.0d, 70.0d));
    }

    @Test public void addProjectProjectNameEmpty(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        //Blank name
        assertThrows(IllegalArgumentException.class, () -> bsf.addProject("", "Bob
(Your uncle)", 50.0d, 70.0d));
    }

    @Test public void addProjectClientNameEmpty(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        //Blank client
        assertThrows(IllegalArgumentException.class, () -> bsf.addProject("Jeff",
"", 50.0d, 70.0d));
    }

    @Test public void addProjectClientNameNull(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        //Null client
        assertThrows(IllegalArgumentException.class, () -> bsf.addProject("Jeff",
null, 50.0d, 70.0d));
    }

    @Test public void addProjectOutOfBoundsRates(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
```

```
        //Under rates
        //Bad Standard Rate (below lower bound)
        assertThrows(IllegalArgumentException.class, () -> bsf.addProject("Jeff",
"e", -5.0d, 70.0d));
        assertThrows(IllegalArgumentException.class, () -> bsf.addProject("Jeff",
"e", 0.0d, 70.0d));

        //Bad Standard Rate (bound, its exclusive)
        assertThrows(IllegalArgumentException.class, () -> bsf.addProject("Jeff",
"e", 0.01d, 70.0d));

        //Bad Standard Rate (upper bound)
        assertThrows(IllegalArgumentException.class, () -> bsf.addProject("Jeff",
"e", 100.0d, 70.0d));

        //Bad Standard Rate (above upper bound)
        assertThrows(IllegalArgumentException.class, () -> bsf.addProject("Jeff",
"e", 101.0d, 70.0d));

        //Over rates
        //Bad Over Rate (below lower bound)
        assertThrows(IllegalArgumentException.class, () -> bsf.addProject("Jeff",
"e", 5.0d, 0.0d));
        assertThrows(IllegalArgumentException.class, () -> bsf.addProject("Jeff",
"e", 5.0d, -5.0d));

        //Bad over rate, on the lower bound
        assertThrows(IllegalArgumentException.class, () -> bsf.addProject("Jeff",
"e", 5.0d, 0.01d));

        //Bad over rate, on the upper bound
        assertThrows(IllegalArgumentException.class, () -> bsf.addProject("Jeff",
"e", 5.0d, 100.0d));

        //bad over rate, above upper bound
        assertThrows(IllegalArgumentException.class, () -> bsf.addProject("Jeff",
"e", 5.0d, 101.0d));
    }

    @Test public void addProjectBadOverrateTenPercent(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        //Bad Overates with 10% bound
        assertThrows(IllegalArgumentException.class, () -> bsf.addProject("Jeff",
"e", 5.0d, 5.1d));
        assertThrows(IllegalArgumentException.class, () -> bsf.addProject("Jeff",
"e", 50.0d, 54.1d));
        assertThrows(IllegalArgumentException.class, () -> bsf.addProject("Jeff",
"e", 95.0d, 110.0d));
    }

    //Valid cases
    @Test
    public void addProjectValidInputs(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
```

```
        Project myProject1 = mock(Project.class);
        Project myProject2 = mock(Project.class);
        Project myProject3 = mock(Project.class);
        Project myProject4 = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), eq("name"), anyDouble(),
anyDouble())).thenReturn(myProject1);
            mock.when(() -> Project.makeProject(anyInt(), eq("name1"), anyDouble(),
anyDouble())).thenReturn(myProject2);
            mock.when(() -> Project.makeProject(anyInt(), eq("name_-"),
anyDouble(), anyDouble())).thenReturn(myProject3);
            mock.when(() -> Project.makeProject(anyInt(), eq("name123__  e"),
anyDouble(), anyDouble())).thenReturn(myProject4);

            assertEquals(bsf.addProject("name", "client", 50.0d, 70.0d),
myProject1);
            assertEquals(bsf.addProject("name1", "cli ent", 1d, 2d), myProject2);
            assertEquals(bsf.addProject("name_-", "clieneweadt", 5d, 10.0d),
myProject3);
            assertEquals(bsf.addProject("name123__  e", "client3", 90.0d, 99.9d),
myProject4);
        }


    }

    @Test public void advBugAddProjectBasicUserTest(){
        when(authenticationModule.login("basicUser",
"password")).thenReturn(basicAuthToken);
        when(authenticationModule.authenticate(basicAuthToken)).thenReturn(true);
        when(authorisationModule.authorise(basicAuthToken,
false)).thenReturn(true);
        //when(authorisationModule.authorise(basicAuthToken,
true)).thenReturn(false);
        bsf.injectAuth(authenticationModule, authorisationModule);
        bsf.login("basicUser", "password");


        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            assertEquals(bsf.addProject("name", "client", 50.0d, 70.0d),
myProject);
            assertEquals(bsf.addProject("name1", "cli ent", 1d, 2d), myProject);
            assertEquals(bsf.addProject("name_-", "clieneweadt", 5d, 10.0d),
myProject);
            assertEquals(bsf.addProject("name123__  e", "client3", 90.0d, 99.9d),
myProject);
        }
    }


    @Test public void advBugAddProjectSecureUserTest(){
        when(authenticationModule.login("secureUser",
"password")).thenReturn(secureAuthToken);
        when(authenticationModule.authenticate(secureAuthToken)).thenReturn(true);
        when(authorisationModule.authorise(secureAuthToken,
```

```
false)).thenReturn(false);
        bsf.injectAuth(authenticationModule, authorisationModule);
        bsf.login("secureUser", "password");

        assertThrows(IllegalStateException.class, () -> bsf.addProject("bob", "s
your uncle", 5.0d, 50.0d));
    }


    ////////////////////////////////////////////////////////////////////
    // removeProject
    ////////////////////////////////////////////////////////////////////
    @Test public void removeProjectNoPermissionsModule(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");

        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            lenient().when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }
        bsf.logout();
        bsf.injectAuth(null, null);

        assertThrows(IllegalStateException.class, () -> bsf.removeProject(1));
    }

    @Test public void removeProjectNoUserLoggedIn(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");

        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            lenient().when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }
        bsf.logout();
        bsf.injectAuth(null, null);

        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        assertThrows(IllegalStateException.class, () -> bsf.removeProject(1));
    }

    @Test public void removeProjectNoProjectID(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");

        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
```

```
            lenient().when(myProject.getId()).thenReturn(0);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }
        bsf.logout();
        bsf.injectAuth(null, null);

        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        assertThrows(IllegalStateException.class, () -> bsf.removeProject(1));
    }

    @Test
    public void removeProjectValid() {
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");

        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }
        assertTrue(bsf.getAllProjects().contains(myProject));
        bsf.removeProject(1);
        assertFalse(bsf.getAllProjects().contains(myProject));
    }

    @Test
    public void removeProjectLargerCase(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");

        Project myProject1 = mock(Project.class);
        Project myProject2 = mock(Project.class);
        Project myProject3 = mock(Project.class);
        Project myProject4 = mock(Project.class);
        Project myProject5 = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), eq("name1"), anyDouble(),
anyDouble())).thenReturn(myProject1);
            mock.when(() -> Project.makeProject(anyInt(), eq("name2"), anyDouble(),
anyDouble())).thenReturn(myProject2);
            mock.when(() -> Project.makeProject(anyInt(), eq("name3"), anyDouble(),
anyDouble())).thenReturn(myProject3);
            mock.when(() -> Project.makeProject(anyInt(), eq("name4"), anyDouble(),
anyDouble())).thenReturn(myProject4);
            mock.when(() -> Project.makeProject(anyInt(), eq("name5"), anyDouble(),
anyDouble())).thenReturn(myProject5);
            bsf.addProject("name1", "client1", 50.0d, 70.0d);
            bsf.addProject("name2", "client2", 50.0d, 70.0d);
            bsf.addProject("name3", "client3", 50.0d, 70.0d);
            bsf.addProject("name4", "client1", 50.0d, 70.0d);
            bsf.addProject("name5", "client1", 50.0d, 70.0d);

            lenient().when(myProject1.getId()).thenReturn(1);
```

```
        lenient().when(myProject2.getId()).thenReturn(2);
        lenient().when(myProject3.getId()).thenReturn(3);
        lenient().when(myProject4.getId()).thenReturn(4);
        lenient().when(myProject5.getId()).thenReturn(5);

        lenient().when(myProject1.getName()).thenReturn("name1");
        lenient().when(myProject2.getName()).thenReturn("name2");
        lenient().when(myProject3.getName()).thenReturn("name3");
        lenient().when(myProject4.getName()).thenReturn("name4");
        lenient().when(myProject5.getName()).thenReturn("name5");
    }

    //having tasks should not effect this at all
    assertTrue(bsf.addTask(1, "Steal the moon", 3, false));
    assertTrue(bsf.addTask(1, "Steal the moon AGAIN", 3, false));
    assertTrue(bsf.addTask(2, "Steal the moon", 3, false));
    assertTrue(bsf.addTask(5, "Steal the moon", 3, false));

    //initially contains all 5
    List<Project> list = bsf.getAllProjects();
    List<Project> client1List = bsf.searchProjects("client1");
    List<Project> client2List = bsf.searchProjects("client2");
    List<Project> client3List = bsf.searchProjects("client3");

    assertTrue(list.contains(myProject1));
    assertTrue(list.contains(myProject2));
    assertTrue(list.contains(myProject3));
    assertTrue(list.contains(myProject4));
    assertTrue(list.contains(myProject5));

    assertTrue(client1List.contains(myProject1));
    assertTrue(client1List.contains(myProject4));
    assertTrue(client1List.contains(myProject5));
    assertTrue(client2List.contains(myProject2));
    assertTrue(client3List.contains(myProject3));

    //Now, remove projects
    bsf.removeProject(2);
    bsf.removeProject(1);
    assertThrows(IllegalStateException.class, () -> bsf.removeProject(1));
    bsf.removeProject(5);

    list = bsf.getAllProjects();
    client1List = bsf.searchProjects("client1");
    client2List = bsf.searchProjects("client2");
    client3List = bsf.searchProjects("client3");

    assertFalse(list.contains(myProject1));
    assertFalse(list.contains(myProject2));
    assertTrue(list.contains(myProject3));
    assertTrue(list.contains(myProject4));
    assertFalse(list.contains(myProject5));

    assertFalse(client1List.contains(myProject1));
    assertTrue(client1List.contains(myProject4));
    assertFalse(client1List.contains(myProject5));
    assertFalse(client2List.contains(myProject2));
    assertTrue(client3List.contains(myProject3));
}
```

```
    @Test public void advBugRemoveProjectBasicUserTest(){
        //First add a project (with valid erp stuff)
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");

        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }
        bsf.logout();
        bsf.injectAuth(null, null);

        //Now log in as basic
        when(authenticationModule.login("basicUser",
"password")).thenReturn(basicAuthToken);
        when(authenticationModule.authenticate(basicAuthToken)).thenReturn(true);
        //when(authorisationModule.authorise(basicAuthToken,
false)).thenReturn(true);
        when(authorisationModule.authorise(basicAuthToken,
true)).thenReturn(false);
        bsf.injectAuth(authenticationModule, authorisationModule);
        bsf.login("basicUser", "password");

        //And assert we cant remove the thing
        assertThrows(IllegalStateException.class, () -> bsf.removeProject(1));
    }

    @Test public void advBugRemoveProjectSecureUserTest(){
        //First add a project (with valid erp stuff)
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");

        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }
        bsf.logout();
        bsf.injectAuth(null, null);

        //Now log in as secure
        when(authenticationModule.login("secureUser",
"password")).thenReturn(secureAuthToken);
        when(authenticationModule.authenticate(secureAuthToken)).thenReturn(true);
        when(authorisationModule.authorise(secureAuthToken,
true)).thenReturn(true);
        bsf.injectAuth(authenticationModule, authorisationModule);
        bsf.login("secureUser", "password");

        //And assert we can remove project
        assertTrue(bsf.getAllProjects().contains(myProject));
        bsf.removeProject(1);
```

```
        assertFalse(bsf.getAllProjects().contains(myProject));
    }


    ////////////////////////////////////////////////////////////////////
    // Add Task
    ////////////////////////////////////////////////////////////////////
    @Test public void addTaskBadProjectID(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");

        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            lenient().when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }

        //Note: ID should not exist
        assertThrows(IllegalStateException.class, () -> bsf.addTask(69, "desc", 10,
false));
    }

    @Test public void addTaskEmptyTaskDescription(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");

        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            lenient().when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }

        //empty task desc
        assertThrows(IllegalArgumentException.class, () -> bsf.addTask(1, "", 10,
false));
    }

    @Test public void addTaskNullTaskDescription(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");

        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            lenient().when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }

        //null task description
        assertThrows(IllegalArgumentException.class, () -> bsf.addTask(1, null, 10,
false));
```

```
    }

    @Test public void addTaskBadHours(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");

        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            lenient().when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }

        //Out of bounds task hours
        assertThrows(IllegalArgumentException.class, () -> bsf.addTask(1, "Steal
the moon", 0, false));
        assertThrows(IllegalArgumentException.class, () -> bsf.addTask(1, "Steal
the moon", -10, false));
        assertThrows(IllegalArgumentException.class, () -> bsf.addTask(1, "Steal
the moon", 1000, false));
    }

    @Test public void addTaskNoPermsMod(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");

        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            lenient().when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }
        bsf.logout();
        bsf.injectAuth(null, null);

        //No permissions package
        assertThrows(IllegalStateException.class, () -> bsf.addTask(1, "moon", 10,
false));
    }

    @Test public void addTaskNoUserLoggedIn(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");

        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            lenient().when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }
        bsf.logout();
        bsf.injectAuth(null, null);
```

```
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        assertThrows(IllegalStateException.class, () -> bsf.addTask(1, "moon", 10,
false));
    }

    @Test public void validAddTasks(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");

        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }
        assertTrue(bsf.addTask(1, "cool description", 10, false));
        assertTrue(bsf.addTask(1, "cool description", 50, false));
        assertFalse(bsf.addTask(1, "cool description", 60, false));
        assertTrue(bsf.addTask(1, "cool description", 40, false));
        assertFalse(bsf.addTask(1, "cool description", 1, false));
        assertTrue(bsf.addTask(1, "cool description", 60, true));
    }

    @Test
    public void addTaskMultipleProjects(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");

        Project myProject1 = mock(Project.class);
        Project myProject2 = mock(Project.class);
        Project myProject3 = mock(Project.class);
        Project myProject4 = mock(Project.class);
        Project myProject5 = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), eq("name1"), anyDouble(),
anyDouble())).thenReturn(myProject1);
            mock.when(() -> Project.makeProject(anyInt(), eq("name2"), anyDouble(),
anyDouble())).thenReturn(myProject2);
            mock.when(() -> Project.makeProject(anyInt(), eq("name3"), anyDouble(),
anyDouble())).thenReturn(myProject3);
            mock.when(() -> Project.makeProject(anyInt(), eq("name4"), anyDouble(),
anyDouble())).thenReturn(myProject4);
            mock.when(() -> Project.makeProject(anyInt(), eq("name5"), anyDouble(),
anyDouble())).thenReturn(myProject5);
            bsf.addProject("name1", "client1", 50.0d, 70.0d);
            bsf.addProject("name2", "client2", 50.0d, 70.0d);
            bsf.addProject("name3", "client3", 50.0d, 70.0d);
            bsf.addProject("name4", "client1", 50.0d, 70.0d);
            bsf.addProject("name5", "client1", 50.0d, 70.0d);
            mock.when(() -> myProject1.getId()).thenReturn(1);
            mock.when(() -> myProject2.getId()).thenReturn(2);
            mock.when(() -> myProject3.getId()).thenReturn(3);
            mock.when(() -> myProject4.getId()).thenReturn(4);
            mock.when(() -> myProject5.getId()).thenReturn(5);
        }
```

```
        assertTrue(bsf.addTask(1, "coolguy", 10, false));
        assertTrue(bsf.addTask(2, "coolguy", 10, true));
        assertTrue(bsf.addTask(1, "coolguy", 90, false));

        assertTrue(bsf.addTask(3, "coolguy", 100, false));
        assertFalse(bsf.addTask(3, "coolguy", 100, false));
        assertTrue(bsf.addTask(3, "coolguy", 100, true));


        assertTrue(bsf.addTask(4, "coolguy", 75, false));
        assertFalse(bsf.addTask(4, "coolguy", 75, false));
        bsf.setProjectCeiling(4, 150);
        assertTrue(bsf.addTask(4, "coolguy2", 75, false));

        assertTrue(bsf.addTask(5, "coolguy", 75, false));
        assertTrue(bsf.addTask(5, "coolguy", 1, false));
        bsf.setProjectCeiling(5, 50);
        assertFalse(bsf.addTask(5, "coolguy", 1, false));
    }

    ////////////////////////////////////////////////////////////////////
    // setProjectCeiling
    ////////////////////////////////////////////////////////////////////
    @Test public void setProjectCeilingNoID(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            lenient().when(myProject.getId()).thenReturn(0);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }

        assertThrows(IllegalStateException.class, () -> bsf.setProjectCeiling(1,
500));
    }

    @Test public void setProjectBadCeiling(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            lenient().when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }

        assertThrows(IllegalArgumentException.class, () -> bsf.setProjectCeiling(1,
-1));
        assertThrows(IllegalArgumentException.class, () -> bsf.setProjectCeiling(1,
0));
        assertThrows(IllegalArgumentException.class, () -> bsf.setProjectCeiling(1,
1001));
        assertThrows(IllegalArgumentException.class, () -> bsf.setProjectCeiling(1,
```

```
100000));
    }

    @Test public void setProjectNoPermsMod(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            lenient().when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }
        bsf.logout();
        bsf.injectAuth(null, null);

        assertThrows(IllegalStateException.class, () -> bsf.setProjectCeiling(1,
500));
    }

    @Test public void setProjectNoUser(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            lenient().when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }
        bsf.logout();
        bsf.injectAuth(null, null);

        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        assertThrows(IllegalStateException.class, () -> bsf.setProjectCeiling(1,
500));
    }

    @Test public void setValidCeilings(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }
        bsf.setProjectCeiling(1, 1);
        bsf.setProjectCeiling(1, 1000);
        bsf.setProjectCeiling(1, 899);
        bsf.setProjectCeiling(1, 100);


        assertTrue(bsf.addTask(1, "cool description", 10, false));
```

```
        assertTrue(bsf.addTask(1, "cool description", 50, false));
        assertFalse(bsf.addTask(1, "cool description", 60, false));
        bsf.setProjectCeiling(1, 300);
        assertTrue(bsf.addTask(1, "cool description", 50, false));
        bsf.setProjectCeiling(1, 30);
        assertFalse(bsf.addTask(1, "cool description", 50, false));
    }

    @Test public void advBugsSetProjectCeiling(){
        bsf.injectAuth(authenticationModule, authorisationModule);
        lenient().when(authenticationModule.login("basicUser",
"password")).thenReturn(basicAuthToken);

lenient().when(authenticationModule.authenticate(basicAuthToken)).thenReturn(true);
        lenient().when(authorisationModule.authorise(basicAuthToken,
false)).thenReturn(true);
        lenient().when(authorisationModule.authorise(basicAuthToken,
true)).thenReturn(false);
        bsf.login("basicUser", "password");

        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            lenient().when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }

        assertThrows(IllegalStateException.class, () -> bsf.setProjectCeiling(1,
500));
    }

    ////////////////////////////////////////////////////////////////////
    // findProjectID
    ////////////////////////////////////////////////////////////////////
    @Test public void findProjectIDNullSearchName(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            lenient().when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }

        assertThrows(IllegalArgumentException.class, () -> bsf.findProjectID(null,
"client"));
    }

    @Test public void findProjectNullClient(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
```

```
            lenient().when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }

        assertThrows(IllegalArgumentException.class, () -> bsf.findProjectID("bob
inc", null));
    }

    @Test public void findProjectIDNoMatchingProject(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            lenient().when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
            lenient().when(myProject.getName()).thenReturn("name");
        }

        assertThrows(IllegalStateException.class, () -> bsf.findProjectID("I dont
exist", "I also dont exist"));
    }

    @Test public void findProjectIDNoMatchingClient(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            lenient().when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
            lenient().when(myProject.getName()).thenReturn("name");
        }

        assertThrows(IllegalStateException.class, () -> bsf.findProjectID("name",
"I also dont exist"));
    }

    @Test public void findProjectIDNoMatchingName(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            lenient().when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
            lenient().when(myProject.getName()).thenReturn("name");
        }

        assertThrows(IllegalStateException.class, () ->
bsf.findProjectID("namen't", "client"));
    }
```

```
    @Test public void addValidProjects(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            when(myProject.getId()).thenReturn(1);
            when(myProject.getName()).thenReturn("name");
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }
        assertEquals(1, bsf.findProjectID("name", "client"));
        assertThrows(IllegalStateException.class, () -> bsf.findProjectID("I dont
exist", "I also dont exist"));
        assertThrows(IllegalStateException.class, () -> bsf.findProjectID("name ",
"client"));
        assertThrows(IllegalStateException.class, () -> bsf.findProjectID("name", "
client"));
        assertThrows(IllegalStateException.class, () -> bsf.findProjectID("namE",
"client"));
        assertThrows(IllegalStateException.class, () -> bsf.findProjectID("name",
"Client"));
    }

    @Test
    public void findProjectIDNotExactCase(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            lenient().when(myProject.getId()).thenReturn(1);
            lenient().when(myProject.getName()).thenReturn("name");
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }
        assertThrows(IllegalStateException.class, () -> bsf.findProjectID("name",
"clienT"));
        assertThrows(IllegalStateException.class, () -> bsf.findProjectID("namE",
"client"));
        assertThrows(IllegalStateException.class, () -> bsf.findProjectID("namE",
"Client"));

        assertThrows(IllegalStateException.class, () -> bsf.findProjectID("name ",
"client"));
        assertThrows(IllegalStateException.class, () -> bsf.findProjectID("name", "
client"));
        assertThrows(IllegalStateException.class, () -> bsf.findProjectID("name ",
" client"));
        assertThrows(IllegalStateException.class, () -> bsf.findProjectID("na me",
"cli ent"));
    }

    @Test
    public void findProjectIDMultipleProjects(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
```

```
        bsf.login("user", "password");

        Project myProject1 = mock(Project.class);
        Project myProject2 = mock(Project.class);
        Project myProject3 = mock(Project.class);
        Project myProject4 = mock(Project.class);
        Project myProject5 = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), eq("name1"), anyDouble(),
anyDouble())).thenReturn(myProject1);
            mock.when(() -> Project.makeProject(anyInt(), eq("name2"), anyDouble(),
anyDouble())).thenReturn(myProject2);
            mock.when(() -> Project.makeProject(anyInt(), eq("name3"), anyDouble(),
anyDouble())).thenReturn(myProject3);
            mock.when(() -> Project.makeProject(anyInt(), eq("name4"), anyDouble(),
anyDouble())).thenReturn(myProject4);
            mock.when(() -> Project.makeProject(anyInt(), eq("name5"), anyDouble(),
anyDouble())).thenReturn(myProject5);

            assertEquals(bsf.addProject("name1", "client1", 50.0d, 70.0d),
myProject1);
            assertEquals(bsf.addProject("name2", "client2", 50.0d, 70.0d),
myProject2);
            assertEquals(bsf.addProject("name3", "client3", 50.0d, 70.0d),
myProject3);
            assertEquals(bsf.addProject("name4", "client1", 50.0d, 70.0d),
myProject4);
            assertEquals(bsf.addProject("name5", "client1", 50.0d, 70.0d),
myProject5);

            lenient().when(myProject1.getId()).thenReturn(1);
            lenient().when(myProject2.getId()).thenReturn(2);
            lenient().when(myProject3.getId()).thenReturn(3);
            lenient().when(myProject4.getId()).thenReturn(4);
            lenient().when(myProject5.getId()).thenReturn(5);

            lenient().when(myProject1.getName()).thenReturn("name1");
            lenient().when(myProject2.getName()).thenReturn("name2");
            lenient().when(myProject3.getName()).thenReturn("name3");
            lenient().when(myProject4.getName()).thenReturn("name4");
            lenient().when(myProject5.getName()).thenReturn("name5");
        }

        assertEquals(bsf.findProjectID("name1", "client1"), 1);
        assertEquals(bsf.findProjectID("name2", "client2"), 2);
        assertEquals(bsf.findProjectID("name3", "client3"), 3);
        assertEquals(bsf.findProjectID("name4", "client1"), 4);
        assertEquals(bsf.findProjectID("name5", "client1"), 5);

        bsf.removeProject(1);
        assertThrows(IllegalStateException.class, () -> bsf.findProjectID("name1",
"client1"));

        //bad combos
        assertThrows(IllegalStateException.class, () -> bsf.findProjectID("name2",
"client3"));
        assertThrows(IllegalStateException.class, () -> bsf.findProjectID("name5",
"client2"));
```

```
    }

    @Test public void findProjectIDBlankNameAndClient(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            lenient().when(myProject.getId()).thenReturn(1);
            lenient().when(myProject.getName()).thenReturn("name");
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }

        assertThrows(IllegalStateException.class, () -> bsf.findProjectID("", ""));
    }

    ////////////////////////////////////////////////////////////////////
    // searchProjects
    ////////////////////////////////////////////////////////////////////
    @Test public void searchProjectsNullClient(){
        assertThrows(IllegalArgumentException.class, () ->
bsf.searchProjects(null));
    }

    @Test public void searchProjectTest1(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        Project myProject = mock(Project.class);
        Project myProject2 = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), eq("name1"), anyDouble(),
anyDouble())).thenReturn(myProject);
            mock.when(() -> Project.makeProject(anyInt(), eq("name2"), anyDouble(),
anyDouble())).thenReturn(myProject2);
            bsf.addProject("name1", "client1", 50.0d, 70.0d);
            bsf.addProject("name2", "client2", 50.0d, 70.0d);
        }

        assertTrue(bsf.searchProjects("client1").contains(myProject));
        assertTrue(bsf.searchProjects("client2").contains(myProject2));
        assertFalse(bsf.searchProjects("client1").contains(myProject2));
        assertFalse(bsf.searchProjects("client2").contains(myProject));
    }

    @Test public void searchProjectDuplicateClients(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");

        Project myProject1 = mock(Project.class);
        Project myProject2 = mock(Project.class);
        Project myProject3 = mock(Project.class);
        Project myProject4 = mock(Project.class);
        Project myProject5 = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), eq("name1"), anyDouble(),
```

```
anyDouble())).thenReturn(myProject1);
            mock.when(() -> Project.makeProject(anyInt(), eq("name2"), anyDouble(),
anyDouble())).thenReturn(myProject2);
            mock.when(() -> Project.makeProject(anyInt(), eq("name3"), anyDouble(),
anyDouble())).thenReturn(myProject3);
            mock.when(() -> Project.makeProject(anyInt(), eq("name4"), anyDouble(),
anyDouble())).thenReturn(myProject4);
            mock.when(() -> Project.makeProject(anyInt(), eq("name5"), anyDouble(),
anyDouble())).thenReturn(myProject5);
            bsf.addProject("name1", "client1", 50.0d, 70.0d);
            bsf.addProject("name2", "client2", 50.0d, 70.0d);
            bsf.addProject("name3", "client3", 50.0d, 70.0d);
            bsf.addProject("name4", "client1", 50.0d, 70.0d);
            bsf.addProject("name5", "client1", 50.0d, 70.0d);

            lenient().when(myProject1.getId()).thenReturn(1);
            lenient().when(myProject2.getId()).thenReturn(2);
            lenient().when(myProject3.getId()).thenReturn(3);
            lenient().when(myProject4.getId()).thenReturn(4);
            lenient().when(myProject5.getId()).thenReturn(5);

            lenient().when(myProject1.getName()).thenReturn("name1");
            lenient().when(myProject2.getName()).thenReturn("name2");
            lenient().when(myProject3.getName()).thenReturn("name3");
            lenient().when(myProject4.getName()).thenReturn("name4");
            lenient().when(myProject5.getName()).thenReturn("name5");
        }

        List<Project> list = bsf.searchProjects("client1");
        assertTrue(list.contains(myProject1));
        assertTrue(list.contains(myProject4));
        assertTrue(list.contains(myProject5));

        assertFalse(list.contains(myProject2));
        assertFalse(list.contains(myProject3));

        List<Project> list2 = bsf.searchProjects("clieNt1");
        assertFalse(list2.contains(myProject1));
        assertFalse(list2.contains(myProject4));
        assertFalse(list2.contains(myProject5));
        assertFalse(list2.contains(myProject2));
        assertFalse(list2.contains(myProject3));

        List<Project> list3 = bsf.searchProjects("client1 ");
        assertFalse(list3.contains(myProject1));
        assertFalse(list3.contains(myProject4));
        assertFalse(list3.contains(myProject5));
        assertFalse(list3.contains(myProject2));
        assertFalse(list3.contains(myProject3));
    }

    ////////////////////////////////////////////////////////////////////
    // getAllProjects
    ////////////////////////////////////////////////////////////////////
    @Test public void getAllProjectsEmptyListNotNull(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
```

```
        assertNotNull(bsf.getAllProjects());
    }

    @Test public void getAllProjects1(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        Project myProject = mock(Project.class);
        Project myProject2 = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), eq("name1"), anyDouble(),
anyDouble())).thenReturn(myProject);
            mock.when(() -> Project.makeProject(anyInt(), eq("name2"), anyDouble(),
anyDouble())).thenReturn(myProject2);
            bsf.addProject("name1", "client", 50.0d, 70.0d);
            bsf.addProject("name2", "client2", 50.0d, 70.0d);
            mock.when(() -> myProject.getId()).thenReturn(1);

        }
        assertTrue(bsf.getAllProjects().contains(myProject));
        assertTrue(bsf.getAllProjects().contains(myProject2));
        bsf.removeProject(1);
        assertFalse(bsf.getAllProjects().contains(myProject));
        assertTrue(bsf.getAllProjects().contains(myProject2));

    }

    ////////////////////////////////////////////////////////////////////
    // audit
    ////////////////////////////////////////////////////////////////////
    @Test public void auditNoPermsMod(){
        assertThrows(IllegalStateException.class, () -> bsf.audit());
    }

    @Test public void auditNoUserLoggedIn(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        assertThrows(IllegalStateException.class, () -> bsf.audit());
    }

    @Test public void auditNoComplianceModSet(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        assertThrows(IllegalStateException.class, () -> bsf.audit());
    }

    @Test public void auditNoProjects(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        bsf.injectCompliance(complianceReporting);
        bsf.audit();
        verify((complianceReporting), times(0)).sendReport(anyString(), anyInt(),
any(AuthToken.class));
    }

    @Test public void auditOneProject(){
        bsf.injectAuth(erp.getAuthenticationModule(),
```

```
erp.getAuthorisationModule());
        bsf.login("user", "password");
        bsf.injectCompliance(complianceReporting);

        Project myProject1 = mock(Project.class);
        Project myProject2 = mock(Project.class);
        Project myProject3 = mock(Project.class);
        Project myProject4 = mock(Project.class);
        Project myProject5 = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), eq("name1"), anyDouble(),
anyDouble())).thenReturn(myProject1);
            mock.when(() -> Project.makeProject(anyInt(), eq("name2"), anyDouble(),
anyDouble())).thenReturn(myProject2);
            mock.when(() -> Project.makeProject(anyInt(), eq("name3"), anyDouble(),
anyDouble())).thenReturn(myProject3);
            mock.when(() -> Project.makeProject(anyInt(), eq("name4"), anyDouble(),
anyDouble())).thenReturn(myProject4);
            mock.when(() -> Project.makeProject(anyInt(), eq("name5"), anyDouble(),
anyDouble())).thenReturn(myProject5);
            bsf.addProject("name1", "client", 50.0d, 70.0d);
            bsf.addProject("name2", "client2", 50.0d, 70.0d);
            bsf.addProject("name3", "client3", 50.0d, 70.0d);
            bsf.addProject("name4", "client4", 50.0d, 70.0d);
            bsf.addProject("name5", "client5", 50.0d, 70.0d);
            lenient().when(myProject1.getId()).thenReturn(1);
            lenient().when(myProject2.getId()).thenReturn(2);
            lenient().when(myProject3.getId()).thenReturn(3);
            lenient().when(myProject4.getId()).thenReturn(4);
            lenient().when(myProject5.getId()).thenReturn(5);

            lenient().when(myProject1.getName()).thenReturn("name1");
            lenient().when(myProject2.getName()).thenReturn("name2");
            lenient().when(myProject3.getName()).thenReturn("name3");
            lenient().when(myProject4.getName()).thenReturn("name4");
            lenient().when(myProject5.getName()).thenReturn("name5");
        }
        //compliant
        assertTrue(bsf.addTask(1, "cool description", 10, false));
        assertTrue(bsf.addTask(2, "cool description", 1, false));
        assertTrue(bsf.addTask(3, "cool description", 100, false));

        //non-compliant
        assertTrue(bsf.addTask(4, "cool description", 50, true));
        assertTrue(bsf.addTask(4, "cool description2", 51, true));
        assertTrue(bsf.addTask(5, "cool description", 75, true));
        assertTrue(bsf.addTask(5, "cool description2", 75, true));

        bsf.audit();
        verify((complianceReporting), times(2)).sendReport(anyString(), anyInt(),
any(AuthToken.class));
        verify((complianceReporting), times(1)).sendReport(anyString(), eq(1),
any(AuthToken.class));
        verify((complianceReporting), times(1)).sendReport(anyString(), eq(50),
any(AuthToken.class));
    }

    @Test public void advBugsAuditAsBasic(){
        bsf.injectAuth(erp.getAuthenticationModule(),
```

```
erp.getAuthorisationModule());
        bsf.login("user", "password");

        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            lenient().when(myProject.getId()).thenReturn(1);
            lenient().when(myProject.getName()).thenReturn("nameLOL");
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }

        bsf.injectCompliance(complianceReporting);
        bsf.addTask(1, "desc", 50, false);
        bsf.addTask(1, "desc", 60, true);

        bsf.logout();
        bsf.injectAuth(null, null);

        //check if we can do this as basic
        bsf.injectAuth(authenticationModule, authorisationModule);
        lenient().when(authenticationModule.login("basicUser",
"password")).thenReturn(basicAuthToken);

lenient().when(authenticationModule.authenticate(basicAuthToken)).thenReturn(true);
        lenient().when(authorisationModule.authorise(basicAuthToken,
true)).thenReturn(false);
        assertTrue(bsf.login("basicUser", "password"));

        assertThrows(IllegalStateException.class, () -> bsf.audit());
    }

    @Test public void auditCorrectToken(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");

        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            lenient().when(myProject.getId()).thenReturn(1);
            lenient().when(myProject.getName()).thenReturn("nameLOL");
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }
        bsf.addTask(1, "desc", 50, false);
        bsf.addTask(1, "desc", 60, true);

        bsf.logout();
        bsf.injectAuth(null,null);

        bsf.injectAuth(authenticationModule, authorisationModule);
        bsf.injectCompliance(complianceReporting);

        lenient().when(authenticationModule.login("secureUser",
"password")).thenReturn(secureAuthToken);

lenient().when(authenticationModule.authenticate(secureAuthToken)).thenReturn(true)
;
```

```
        lenient().when(authorisationModule.authorise(secureAuthToken,
true)).thenReturn(true);
        lenient().when(authorisationModule.authorise(secureAuthToken,
false)).thenReturn(false);
        bsf.login("secureUser", "password");

        bsf.audit();
        verify((complianceReporting), times(1)).sendReport(anyString(), anyInt(),
any(AuthToken.class));
        verify((complianceReporting), times(1)).sendReport(anyString(), eq(10),
eq(secureAuthToken));
    }

    @Test public void auditInteractionsWithChangingProjects(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");

        Project myProject1 = mock(Project.class);
        Project myProject2 = mock(Project.class);
        Project myProject3 = mock(Project.class);
        Project myProject4 = mock(Project.class);
        Project myProject5 = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), eq("name1"), anyDouble(),
anyDouble())).thenReturn(myProject1);
            mock.when(() -> Project.makeProject(anyInt(), eq("name2"), anyDouble(),
anyDouble())).thenReturn(myProject2);
            mock.when(() -> Project.makeProject(anyInt(), eq("name3"), anyDouble(),
anyDouble())).thenReturn(myProject3);
            mock.when(() -> Project.makeProject(anyInt(), eq("name4"), anyDouble(),
anyDouble())).thenReturn(myProject4);
            mock.when(() -> Project.makeProject(anyInt(), eq("name5"), anyDouble(),
anyDouble())).thenReturn(myProject5);
            bsf.addProject("name1", "client", 50.0d, 70.0d);
            bsf.addProject("name2", "client2", 50.0d, 70.0d);
            bsf.addProject("name3", "client3", 50.0d, 70.0d);
            bsf.addProject("name4", "client4", 50.0d, 70.0d);
            bsf.addProject("name5", "client5", 50.0d, 70.0d);

            lenient().when(myProject1.getId()).thenReturn(1);
            lenient().when(myProject2.getId()).thenReturn(2);
            lenient().when(myProject3.getId()).thenReturn(3);
            lenient().when(myProject4.getId()).thenReturn(4);
            lenient().when(myProject5.getId()).thenReturn(5);

            lenient().when(myProject1.getName()).thenReturn("name1");
            lenient().when(myProject2.getName()).thenReturn("name2");
            lenient().when(myProject3.getName()).thenReturn("name3");
            lenient().when(myProject4.getName()).thenReturn("name4");
            lenient().when(myProject5.getName()).thenReturn("name5");
        }
        assertTrue(bsf.addTask(1, "random desc", 10, false));

        assertTrue(bsf.addTask(2, "random desc", 50, true));
        assertTrue(bsf.addTask(2, "random desc", 60, true));

        assertTrue(bsf.addTask(3, "random desc", 100, false));
```

```
        assertTrue(bsf.addTask(4, "random desc", 10, false));
        assertTrue(bsf.addTask(4, "random desc", 10, false));

        bsf.injectCompliance(complianceReporting);

        bsf.audit();
        verify((complianceReporting), times(1)).sendReport(anyString(), anyInt(),
any(AuthToken.class));
        verify((complianceReporting), times(1)).sendReport(anyString(), eq(10),
any(AuthToken.class));

        bsf.setProjectCeiling(2, 50);
        bsf.setProjectCeiling(3, 50);
        bsf.setProjectCeiling(4, 15);

        bsf.audit();
        verify((complianceReporting), times(4)).sendReport(anyString(), anyInt(),
any(AuthToken.class));
        verify((complianceReporting), times(1)).sendReport(anyString(), eq(50),
any(AuthToken.class));
        verify((complianceReporting), times(1)).sendReport(anyString(), eq(60),
any(AuthToken.class));
        verify((complianceReporting), times(1)).sendReport(anyString(), eq(5),
any(AuthToken.class));

        bsf.removeProject(2);
        bsf.injectClient(clientReporting);
        bsf.finaliseProject(3);

        bsf.audit();
        verify((complianceReporting), times(5)).sendReport(anyString(), anyInt(),
any(AuthToken.class));
        verify((complianceReporting), times(2)).sendReport(anyString(), eq(5),
any(AuthToken.class));
    }

    ////////////////////////////////////////////////////////////////////
    // finaliseProject
    ////////////////////////////////////////////////////////////////////
    @Test public void finaliseProjectNoPerms(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            lenient().when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }
        bsf.logout();
        bsf.injectAuth(null, null);

        assertThrows(IllegalStateException.class, () -> bsf.finaliseProject(1));
    }

    @Test public void finaliseProjectNoUser(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
```

```
        bsf.login("user", "password");
        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            lenient().when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }
        bsf.logout();
        bsf.injectAuth(null, null);

        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        assertThrows(IllegalStateException.class, () -> bsf.finaliseProject(1));
    }

    @Test public void finaliseProjectNoID(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            lenient().when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }
        bsf.logout();
        bsf.injectAuth(null, null);

        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        bsf.injectClient(clientReporting);
        assertThrows(IllegalStateException.class, () -> bsf.finaliseProject(69));
    }

    @Test public void finaliseProjectNoClientReporting(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            lenient().when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }
        bsf.logout();
        bsf.injectAuth(null, null);

        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        assertThrows(IllegalStateException.class, () -> bsf.finaliseProject(1));
    }

    @Test public void finaliseProjectTest1(){
        bsf.injectAuth(erp.getAuthenticationModule(),
```

```
erp.getAuthorisationModule());
        bsf.login("user", "password");
        bsf.injectClient(clientReporting);

        Project myProject1 = mock(Project.class);
        Project myProject2 = mock(Project.class);
        Project myProject3 = mock(Project.class);
        Project myProject4 = mock(Project.class);
        Project myProject5 = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), eq("name1"), anyDouble(),
anyDouble())).thenReturn(myProject1);
            mock.when(() -> Project.makeProject(anyInt(), eq("name2"), anyDouble(),
anyDouble())).thenReturn(myProject2);
            mock.when(() -> Project.makeProject(anyInt(), eq("name3"), anyDouble(),
anyDouble())).thenReturn(myProject3);
            mock.when(() -> Project.makeProject(anyInt(), eq("name4"), anyDouble(),
anyDouble())).thenReturn(myProject4);
            mock.when(() -> Project.makeProject(anyInt(), eq("name5"), anyDouble(),
anyDouble())).thenReturn(myProject5);
            bsf.addProject("name1", "client", 50.0d, 70.0d);
            bsf.addProject("name2", "client2", 50.0d, 70.0d);
            bsf.addProject("name3", "client3", 50.0d, 70.0d);
            bsf.addProject("name4", "client4", 50.0d, 70.0d);
            bsf.addProject("name5", "client5", 50.0d, 70.0d);
            mock.when(() -> myProject1.getId()).thenReturn(1);
            mock.when(() -> myProject2.getId()).thenReturn(2);
            mock.when(() -> myProject3.getId()).thenReturn(3);
            mock.when(() -> myProject4.getId()).thenReturn(4);
            mock.when(() -> myProject5.getId()).thenReturn(5);
        }

        assertTrue(bsf.getAllProjects().contains(myProject1));
        bsf.finaliseProject(1);
        verify(clientReporting, times(1)).sendReport(eq("client"), anyString(),
any(AuthToken.class));
        assertFalse(bsf.getAllProjects().contains(myProject1));

        bsf.finaliseProject(2);
        verify(clientReporting, times(1)).sendReport(eq("client2"), anyString(),
any(AuthToken.class));
        assertFalse(bsf.getAllProjects().contains(myProject2));

        bsf.finaliseProject(5);
        verify(clientReporting, times(1)).sendReport(eq("client5"), anyString(),
any(AuthToken.class));
        assertFalse(bsf.getAllProjects().contains(myProject5));
    }

    @Test
    public void advBugFinaliseProject1(){
        //First add a project that works
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");

        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
```

```
anyDouble())).thenReturn(myProject);
            lenient().when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }
        bsf.logout();
        bsf.injectAuth(null, null);

        bsf.injectClient(clientReporting);

        bsf.injectAuth(authenticationModule, authorisationModule);
        lenient().when(authenticationModule.login("basicUser",
"password")).thenReturn(basicAuthToken);

lenient().when(authenticationModule.authenticate(basicAuthToken)).thenReturn(true);
        lenient().when(authorisationModule.authorise(basicAuthToken,
true)).thenReturn(false);
        lenient().when(authorisationModule.authorise(basicAuthToken,
false)).thenReturn(true);
        bsf.login("basicUser", "password");


        assertThrows(IllegalStateException.class, () -> bsf.finaliseProject(1));

        lenient().when(authenticationModule.login("secureUser",
"password")).thenReturn(secureAuthToken);

lenient().when(authenticationModule.authenticate(secureAuthToken)).thenReturn(true)
;
        lenient().when(authorisationModule.authorise(secureAuthToken,
true)).thenReturn(true);
        lenient().when(authorisationModule.authorise(secureAuthToken,
false)).thenReturn(false);
        bsf.login("secureUser", "password");

        assertThrows(IllegalStateException.class, () -> bsf.finaliseProject(1));
    }


    @Test public void advBugsFinaliseProject(){
        //First add a project that works
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");

        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            lenient().when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }
        bsf.logout();
        bsf.injectAuth(null, null);


        bsf.injectClient(clientReporting);

        bsf.injectAuth(authenticationModule, authorisationModule);
        lenient().when(authenticationModule.login("basicUser",
```

```
"password")).thenReturn(basicAuthToken);

lenient().when(authenticationModule.authenticate(basicAuthToken)).thenReturn(true);
        lenient().when(authorisationModule.authorise(basicAuthToken,
true)).thenReturn(false);
        lenient().when(authorisationModule.authorise(basicAuthToken,
false)).thenReturn(true);
        bsf.login("basicUser", "password");


        assertThrows(IllegalStateException.class, () -> bsf.finaliseProject(1));

        lenient().when(authenticationModule.login("secureUser",
"password")).thenReturn(secureAuthToken);

lenient().when(authenticationModule.authenticate(secureAuthToken)).thenReturn(true)
;
        lenient().when(authorisationModule.authorise(secureAuthToken,
true)).thenReturn(true);
        lenient().when(authorisationModule.authorise(secureAuthToken,
false)).thenReturn(false);
        assertTrue(bsf.login("secureUser", "password"));

        assertThrows(IllegalStateException.class, () -> bsf.finaliseProject(1));

        //Now, become doug dimmadome
        lenient().when(authenticationModule.login("basicAndSecureUser",
"password")).thenReturn(basicAndSecure);

lenient().when(authenticationModule.authenticate(basicAndSecure)).thenReturn(true);
        lenient().when(authorisationModule.authorise(basicAndSecure,
true)).thenReturn(true);
        lenient().when(authorisationModule.authorise(basicAndSecure,
false)).thenReturn(true);
        assertTrue(bsf.login("basicAndSecureUser", "password"));

        bsf.finaliseProject(1);
        verify(clientReporting, times(1)).sendReport(anyString(), anyString(),
any(AuthToken.class));
        verify(clientReporting, times(1)).sendReport(eq("client"), anyString(),
eq(basicAndSecure));

        assertFalse(bsf.getAllProjects().contains(myProject));
    }

    ////////////////////////////////////////////////////////////////////
    // injectAuth
    ////////////////////////////////////////////////////////////////////
    @Test public void injectAuthOneNullBad(){
        assertThrows(IllegalArgumentException.class, () -> bsf.injectAuth(null,
erp.getAuthorisationModule()));
        assertThrows(IllegalArgumentException.class, () ->
bsf.injectAuth(erp.getAuthenticationModule(), null));
    }

    @Test public void validAuthNullInjectionTests(){
        bsf.injectAuth(null, null);

        bsf.injectAuth(erp.getAuthenticationModule(),
```

```
erp.getAuthorisationModule());
        bsf.injectAuth(null, null);
        assertThrows(IllegalStateException.class, () -> bsf.audit());

        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");
        bsf.injectAuth(null, null);
        assertThrows(IllegalStateException.class, () -> bsf.audit());
    }

    @Test
    public void injectAuthRemoveAuthWhileLoggedIn(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");

        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            bsf.addProject("name", "client", 50.0d, 70.0d);
            when(myProject.getId()).thenReturn(1);
        }
        bsf.injectAuth(null, null);

        assertThrows(IllegalStateException.class, () ->  bsf.addTask(1, "desc", 3,
false));
    }

    ////////////////////////////////////////////////////////////////////
    // injectClient
    ////////////////////////////////////////////////////////////////////
    //No error cases, play with later
    @Test
    public void testAddingClient(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");

        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }

        bsf.injectClient(clientReporting);
        bsf.injectClient(null);
        assertThrows(IllegalStateException.class, () ->  bsf.finaliseProject(1));
    }

    ////////////////////////////////////////////////////////////////////
    // injectCompliance
    ////////////////////////////////////////////////////////////////////
    //same deal as ^
    @Test
    public void testAddingCompliance(){
```

```
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");

        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            lenient().when(myProject.getId()).thenReturn(1);
            bsf.addProject("name", "client", 50.0d, 70.0d);
        }

        bsf.injectCompliance(complianceReporting);
        bsf.audit();

        bsf.injectCompliance(null);

        assertThrows(IllegalStateException.class, () ->  bsf.audit());
    }

    ////////////////////////////////////////////////////////////////////
    // login
    ////////////////////////////////////////////////////////////////////
    @Test public void loginNullUsername(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        assertThrows(IllegalArgumentException.class, () -> bsf.login(null,
"password"));
    }

    @Test public void loginNullPassword(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        assertThrows(IllegalArgumentException.class, () -> bsf.login("basicUser",
null));
    }

    @Test public void loginNoPermsMod(){
        assertThrows(IllegalStateException.class, () -> bsf.login("basicUser",
"password"));
    }

    @Test public void loginValidCases(){
        //Log in as a secure user and make sure we can do stuff
        AuthToken exampleToken = mock(AuthToken.class);
        when(authenticationModule.login("", "")).thenReturn(exampleToken);
        when(authenticationModule.login("notAUserName",
"notAPassword")).thenReturn(null);

//when(authenticationModule.authenticate(secureAuthToken)).thenReturn(true);
        //when(authorisationModule.authorise(secureAuthToken,
false)).thenReturn(false);
        //when(authorisationModule.authorise(secureAuthToken,
true)).thenReturn(true);
        bsf.injectAuth(authenticationModule, authorisationModule);
        assertTrue(bsf.login("", ""));
        assertFalse(bsf.login("notAUserName", "notAPassword"));
    }
```

```
    @Test public void loginOverWriteExistingUser(){
        //First add a project (with valid erp stuff)
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");

        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            bsf.addProject("name", "client", 50.0d, 70.0d);
            lenient().when(myProject.getId()).thenReturn(1);
            lenient().when(myProject.getName()).thenReturn("name");
        }
        bsf.logout();
        bsf.injectAuth(null, null);
        //End add project

        //Log in as basic
        lenient().when(authenticationModule.login("basicUser",
"password")).thenReturn(basicAuthToken);

lenient().when(authenticationModule.authenticate(basicAuthToken)).thenReturn(true);
        lenient().when(authorisationModule.authorise(basicAuthToken,
false)).thenReturn(true);
        bsf.injectAuth(authenticationModule, authorisationModule);
        assertTrue(bsf.login("basicUser", "password"));

        //Log in as secure
        lenient().when(authenticationModule.login("secureUser",
"password")).thenReturn(secureAuthToken);

lenient().when(authenticationModule.authenticate(basicAuthToken)).thenReturn(false)
;
        lenient().when(authorisationModule.authorise(basicAuthToken,
false)).thenReturn(false);

lenient().when(authenticationModule.authenticate(secureAuthToken)).thenReturn(true)
;
        lenient().when(authorisationModule.authorise(secureAuthToken,
true)).thenReturn(true);
        assertTrue(bsf.login("secureUser", "password"));

        //Make sure we cant do basic stuff and can do secure
        assertThrows(IllegalStateException.class, () -> bsf.addTask(1, "eMan", 3,
false));
        assertTrue(bsf.addTask(1, "eMan", 3, true));

        bsf.logout();

lenient().when(authenticationModule.authenticate(secureAuthToken)).thenReturn(false
);
        lenient().when(authorisationModule.authorise(secureAuthToken,
true)).thenReturn(false);

        assertThrows(IllegalStateException.class, () -> bsf.addTask(1, "eMan", 3,
true));
        assertThrows(IllegalStateException.class, () -> bsf.addTask(1, "eMan", 3,
false));
```

```
    }


    ////////////////////////////////////////////////////////////////////
    // logout
    ////////////////////////////////////////////////////////////////////
    @Test public void logoutNoUser(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        assertThrows(IllegalStateException.class, () -> bsf.logout());
    }

    @Test public void logoutNoPermsMod(){
        assertThrows(IllegalStateException.class, () -> bsf.logout());
    }

    @Test
    public void logoutActuallyLogOut(){
        bsf.injectAuth(erp.getAuthenticationModule(),
erp.getAuthorisationModule());
        bsf.login("user", "password");

        Project myProject = mock(Project.class);
        try (MockedStatic<Project> mock = mockStatic(Project.class)) {
            mock.when(() -> Project.makeProject(anyInt(), anyString(), anyDouble(),
anyDouble())).thenReturn(myProject);
            bsf.addProject("name", "client", 50.0d, 70.0d);
            when(myProject.getId()).thenReturn(1);
        }
        bsf.logout();

        assertThrows(IllegalStateException.class, () ->  bsf.addTask(1, "desc", 3,
false));
    }
}


////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
```