UNIVERSITÄT PASSAU

Implementerungsplan

DocThesisTracker

Client: Dr. Armin Größlinger

Team 2

Hadi Abou Hassoun
Alaa Qasem
Johannes Silvennoinen
Stefan Witka

Inhaltsverzeichnis

1 Einleitung

Johannes Silvennoinen

Dies ist der Implementierungsplan für die folgenden 3 Wochen vom 9.06.2023 bis zum 23.06.2023, je Woche ein Milestone, wobei jedes Milestone so vertikal ist wie möglich bzw. aus kleineren abhängigen und unabhängigen Arbeitspaketen aus jeder Schicht.

Im ersten Milestone werde die wichtigsten Sachen implementiert sowie Login, Connectionpool, ConfigReader, Umlauf-erstellen, Systeminitalizer und Abhängigkeiten für die Umlauferstellung und das einloggen. Das Ziel ist es das der Benutzer sich am ende des Milestones eingoggen kann und einen Umlauferstellen kann sowie den Umlauf anschauen kann. Somit ist das lesen und schreiben in die Datenbank schon einmal implementiert sowie Facelets und Backing Beans.

Im zweiten Milestone werden die restlichen wichtigen Features implementiert, sowie die Fakultäten, die Benutzer-listen und Verwaltung, Registrierung samt Abhängigkeiten wie Tokenmanager und Maintenancethread, sowie letztendlich das abstimmen von Umläufen. Das Ziel des zweiten Milestones ist es das man sich neu Registrieren kann, dass die Fakultäten anwendbar sind und das man die Benutzerverwalten kann.

Im dritten Milestone implementieren wir noch die letzten Features sowie CSS, Fehlerbehandlung und implementieren wenn möglich noch Wunschkriterien sowie Resource Bundles oder Email Sender. Wunschkriterien sind mit einem ”*” in den Arbeitspaketen gekennzeichnet.

Die Bearbeitungszeiten wurden innerhalb vom Team abgeschätzt, wobei der Experte eine stärkere Gewichtung bekommen hat beim abschätzen je Arbeitspaket.

2 Milestone 1

Hadi Abou Hasssuoun

LoginBacking	05.06.2023	06.06.2023	5 h	Alaa Qasem	init() login() register() forgetPass() login.xhtml
SessionInfo	05.06.2023	06.06.2023	1.5 h	Johannes Silvennoinen	isAdmin()
SystemInitializer	05.06.2023	06.06.2023	1.5 h	Johannes Silvennoinen	contextInitialized() contextDestroyed()
ConfigReader	05.06.2023	06.06.2023	2 h	Johannes Silvennoinen	loadProperties() getProperty()
ConnectionPool	05.06.2023	06.06.2023	6 h	Stefan Witka	ConnectionPool: getConnection() releaseConnection() createConnection() initializeConnections() shutdown() Transaction: abort(), commit(), close() getConnection()
Pagination	05.06.2023	06.06.2023	1.5 h	Alaa Qasem	nextPage() previousPage() firstPage() sortBy() calculateNumberOfPages()
Main	05.06.2023	06.06.2023	3 h	Hadi Abou Hassoun	Main.xhtml

Exeptions	05.06.2023	07.06.2023	2 h	Hadi Abou Hassoun	ConfigurationReadException() DataIntegrityException() DataNotCompleteException() DataNotFoundException() DBConnectionFailedException() InvalidInputException() KeyExistsException()
FutureDateTimeValidator	05.06.2023	06.06.2023	2 h	Hadi Abou Hassoun	convertToDateTime() validate()
Hader	05.06.2023	09.06.2023	4 h	Hadi Abou Hassoun	header.xhtml
footer	05.06.2023	06.06.2023	4 h	Hadi Abou Hassoun	footer.xhtml
EmailAddressSyntaxValidator	05.06.2023	9.06.2023	2h	Hadi Abou Hassoun	validate() isEmailAddressAvailable()
Trespasslistener	5.06.2023	9.06.2023	4 h	Johannes Silvennoinen	afterPhase() beforePhase() getPhaseId()
Hashing	05.06.2023	9.06.2023	2 h	Johannes Silvennoinen	hashPassword() verifyPassword()
NavigationBacking	05.06.2023	9.06.2023	2 h	Alaa Qasem	logout()

UserDAO	06.06.2023	07.06.2023	7 h	Stefan Witka	add() remove() update()
ErrorPageBacking	07.06.2023	9.06.2023	2 h	Alaa Qasem	getErrorMessage() setErrorMesage() init() errorpage.xhtml
CirculationCreatingBacking	09.06.2023	09.06.2023	8 h	Stefan Witka	init() create() createCirculation.xhtml
CirculaitonDAO	08.06.2023	08.06.2023	7 h	Johannes Silvennoinen	add() remove() update() getCirculationById() findCirculationByTitle() getTotalCirculationNumber()
CirculaitonList	09.06.2023	09.06.2023	8 h	Hadi Abou Hassoun	loadData() init() setCircPagination() getCircPagination() CirculationList.xhtml
CirculaitonDetails	09.06.2023	09.06.2023	6 h	Alaa Qasem	loadCirculation init() Circulaitondetails.xhtml
Test	08.06.2023	08.06.2023	5 h	Alaa Qasem	testLoginSuccessful() testLoginFailed()
HashingTest	08.06.2023	09.06.2023	3 h	Johannes Silvennoinen	testHashedPassword() testVerifyPasswordMatching() testVerifyPasswordMismatched() testVerifyPasswordIncorrectSalt() testVerifPasswordIncorrectPassword()

Summe die Stunden : 88.5 h

Teammitglied	**stundenAnzahl**
Hadi Abou Hassoun	23
Alaa Qasem	21.5
Johannes Silvennoinen	21
Stefan Witka	21

3 Milestone 2

Stefan Witka

Arbeitspaket	Start	Ende	Aufwand	Bearbeiter	Artefakte
RegistrationBacking	12.06.2023	15.06.2023	5 h	Johannes Silvennoinen	init() register()
UserListBacking	12.06.2023	16.06.2023	5 h	Hadi Abou Hassoun	init()
ProfileBacking	12.06.2023	16.06.2023	5 h	Alaa Qasem	deleteProfile() load() save() viewAction()
EmailAddress Availability	12.06.2023	16.06.2023	3 h	Alaa Qasem	validate() isEmailAdressAvailable
UniqueCirculationName Validator	12.06.2023	16.06.2023	5 h	Stefan Witka	validate() isValueUnique()
TokenManager	12.06.2023	16.06.2023	5 h	Johannes Silvennoinen	lookupToken() generateToken() clearExpiredTokens()
FacultyDAO	12.06.2023	14.06.2023	5 h	Alaa Qasem	FacultyDAO() add() remove() update() getFaculties() findFacultyByName()
VoteDAO	12.06.2023	14.06.2023	5h	Hadi Abou Hassoun	VoteDAO() add() remove() update getVotes() findVote()
Voting functions	14.06.2023	16.06.2023	4h	Stefan Witka	Circulationdetails.xhtml CirculationDetailsBacking: vote() loadVotes()
FacultyBacking	14.06.2023	16.06.2023	5 h	Stefan Witka	init() add() remove()
MaintenanceThread	12.06.2023	16.06.2023	5 h	Johannes Silvennoinen	run() startMaintenace() stopMaintenance
RegistrationBacking Tests	15.06.2023	16.06.2023	5 h	Hadi Abou Hassoun	testRegister_SuccessfulRegistration() testRegister_DuplicateEmail()

Teammitglied	**stundenAnzahl**
Hadi Abou Hassoun	15
Alaa Qasem	13
Johannes Silvennoinen	15
Stefan Witka	14

4 Milestone 3

Alaa Qasem

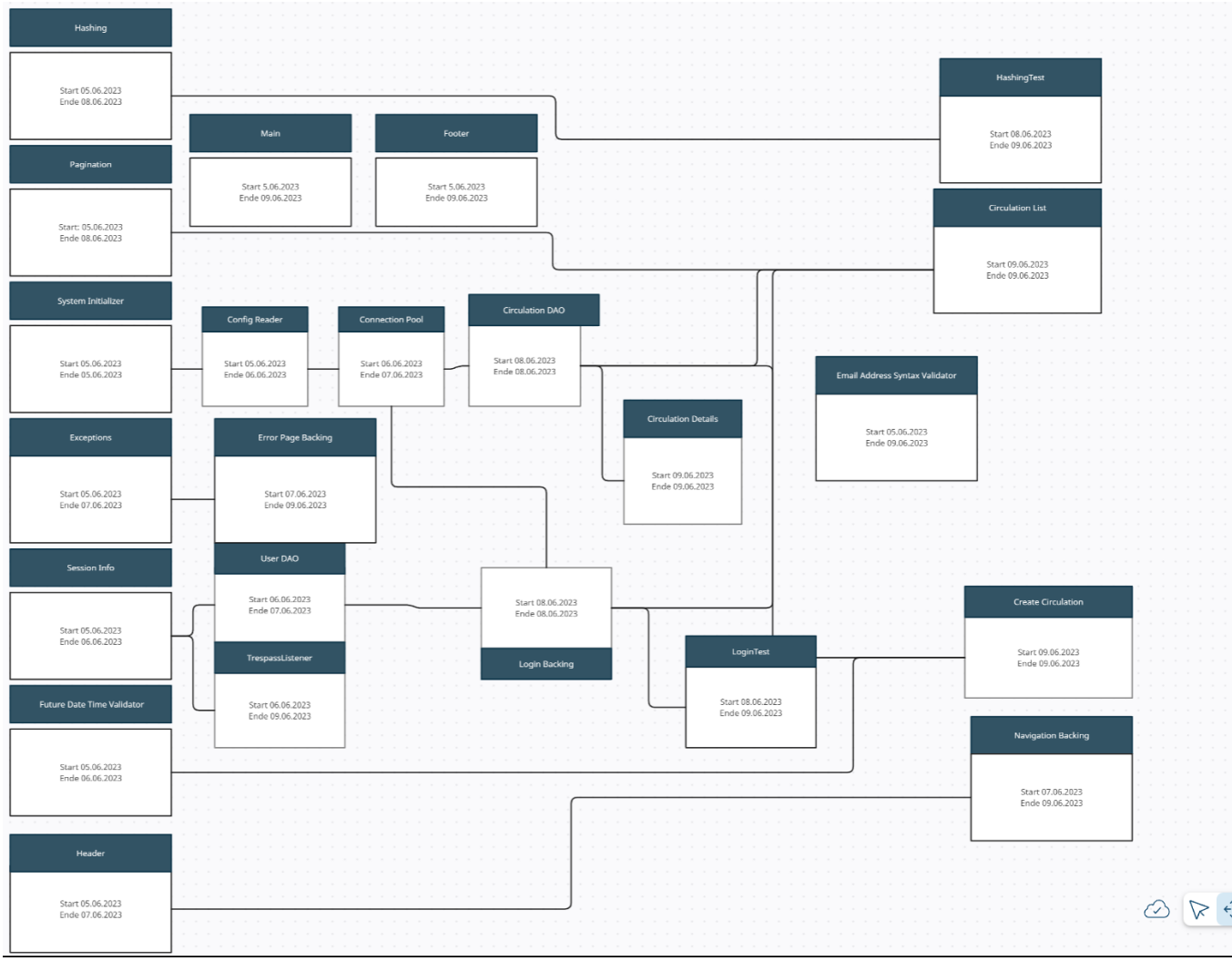
Arbeitspaket	**Start**	**Ende**	**Aufwand**	**Bearbeiter**	**Artefakte**
SetNewPassword Backing	19.06.2023	23.06.2023	5 h	Stefan Witka	save() getUserDAO() setUserDAO() getSessionInfo() setSessionInfo() getPassword
ResetPasswordBacking	19.06.2023	23.06.2023	5 h	Alaa Qasem	sendResetPasswordEmail()
UniqueFacultyName Validator	19.06.2023	22.06.2023	5 h	Hadi Abou Hassoun	validate() isValueUnique()
CSS	19.06.2023	23.06.2023	5 h	Hadi Abou Hassoun	default
UncheckedException	19.06.2023	23.06.2023	5 h	Johannes Silvennoinen	UEHandler: handle() UEHandlerFactory: getExceptionhandler()
* ResourceBundles	19.06.2023	23.06.2023	10 h	Johannes Silvennoinen	ResourceBundleManager Resource Bundles
* EmailSender	19.06.2023	23.06.2023	5 h	Hadi Abou Hassoun	sendEmail()
Handbuch	19.06.2023	23.06.2023	5 h	Alaa Qasem	Help.xhtml * HelpBacking: * goToContext()
UniqueFacultyName-ValidatorTest	22.06.2023	23.06.2023	3 h	Stefan Witka	validateTest isValueUniqueTest

Teammitglied	**stundenAnzahl**
Hadi Abou Hassoun	15
Alaa Qasem	10
Johannes Silvennoinen	15
Stefan Witka	8

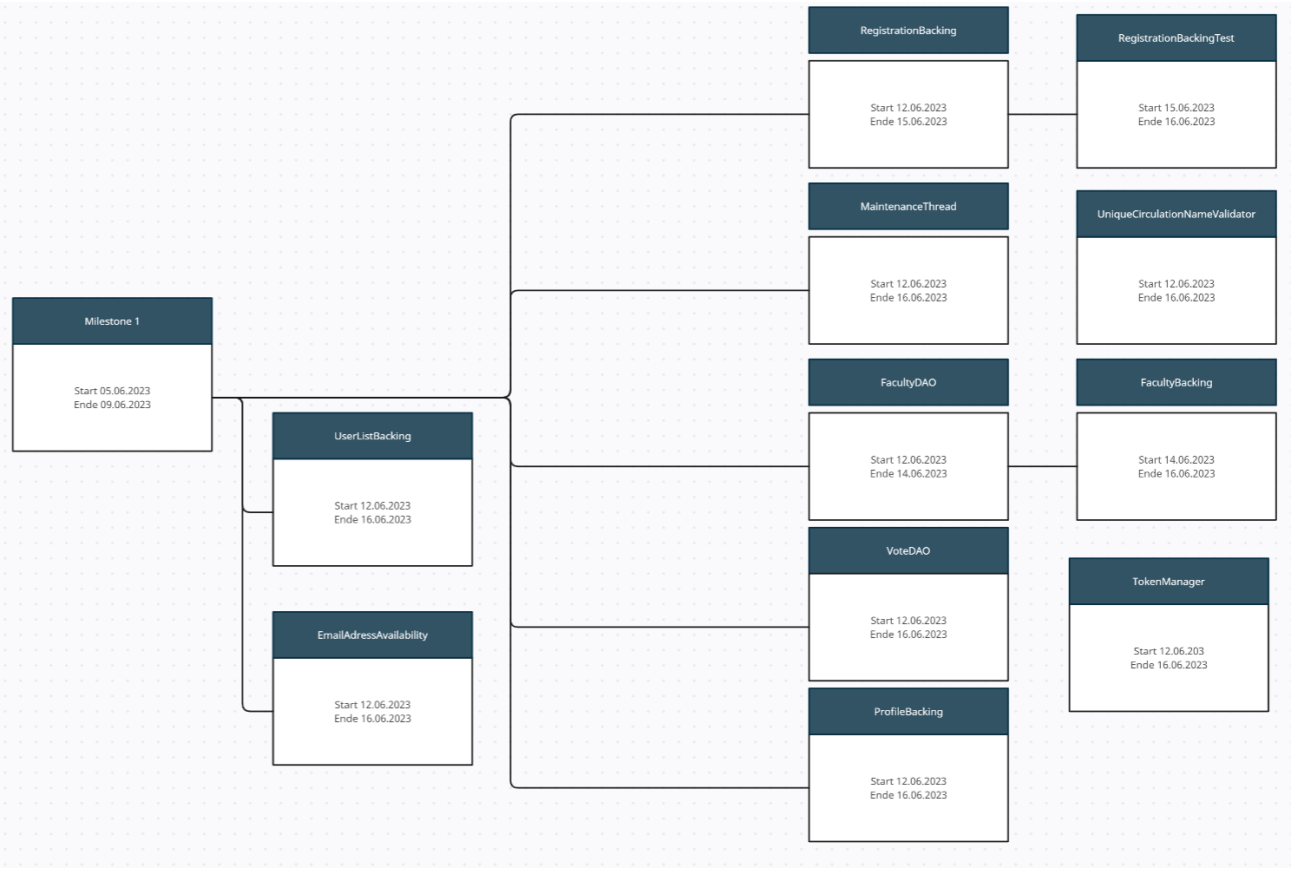
5 Abhängigkeiten

Johannes Silvennoinen

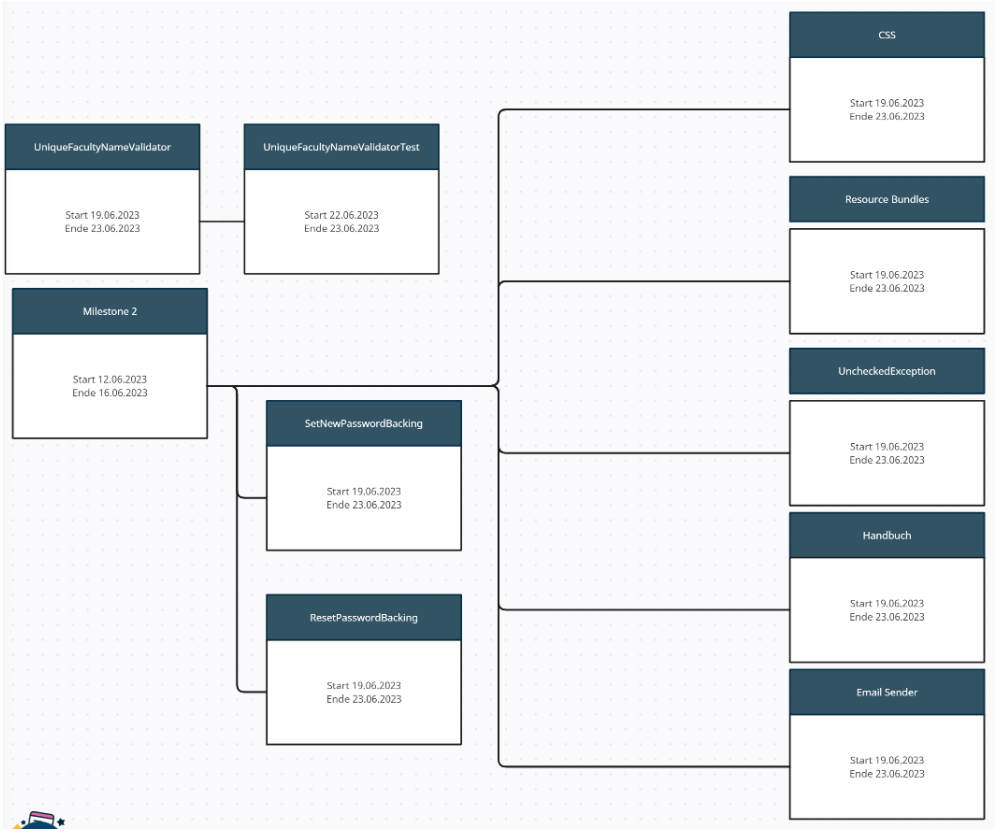
5.1 Milestone 1



5.2 Milestone 2



5.3 Milestone 3



6 Spezialgebiete

Alaa Qasem

Spezialgebiet	**Experte**
Jakarta Server Faces	Hadi Abou Hassoun
CSS	Hadi Abou Hassoun
Logging	Johannes Silvennoinen
SQL	Stefan Witka
Servlets	Alaa Qasem
JUnit	Alaa Qasem
Crypto	Johannes Silvennoinen

7 Tests

7.1 Milestone 1

7.1.1 HashingTest

Johannes Silvennoinen

```
 /**
 * Test case to verify that the hashed password matches the expected hash when
 * the same password and salt are used.
 */
@Test
public void testHashPassword() {
}

/**
 * Test case to verify that the verifyPassword method returns true when the
 * correct password, salt, and hash are provided.
 */
@Test
public void testVerifyPasswordMatching() {
}

/**
 * Test case to verify that the verifyPassword method returns false when an
 * incorrect hash is provided.
 */
@Test
public void testVerifyPasswordMismatched() {
}

/**
 * Test case to verify that the verifyPassword method returns false when an
 * incorrect salt is provided.
 */
@Test
public void testVerifyPasswordIncorrectSalt() {
}

/**
```

```
 * Test case to verify that the verifyPassword method returns false when an
 * incorrect password is provided.
 */
@Test
public void testVerifyPasswordIncorrectPassword() {

}
```

7.1.2 LoginTest

Alaa Qasem

```
 /**
 * This class contains JUnit tests for the LoginBacking class.
 * It verifies the login functionality for successful and failed login attempts.
 */
public class LoginBackingTest {

    /**
     * Test case to verify successful login.
     * It checks whether the login method returns the expected navigation outcome.
     */
    @BeforeEach
    void setUp() {
        // Set up test dependencies and data
    }

    /**
     * Test case to verify successful login.
     * It checks whether the login method returns the expected navigation outcome
     * ("/view/circulationList").
     */
    @Test
    void testLogin_Successful() {
        // Set up test data

        // Call the login method

        // Verify the expected outcome ("/view/circulationList")
    }
```

```
    /**
     * Test case to verify failed login.
     * It checks whether the login method returns null, indicating a failed login
     * attempt.
     */
    @Test
    void testLogin_Failed() {
        // Set up test data

        // Call the login method

        // Verify the expected outcome (null)
    }
}
```

7.2 Milestone 2

Hadi Abou Hassoun

7.2.1 RegistrationBackingTests

```
public class RegistrationBackingTest {

    // Mocks für  UserDAO  FacesContext
    @Mock
    private UserDAO userDAO;

    // Mocks für FacesContext
    @Mock
    private FacesContext facesContext;

    // Die RegistrationBacking-Klasse wird mit den Mocks initialisiert

    @InjectMocks
    private RegistrationBacking registrationBacking;

    @BeforeEach
```

```
public void setup() {
    // Initialisierung der Mocks
    MockitoAnnotations.openMocks(this);
}

@Test
void testRegister_SuccessfulRegistration() {
    // Erzeugung eines Testbenutzers
    User user = new User();
    user.setEmail("test@uni-passau.de");

    // Verhalten des UserDAO-Mocks festlegen:
    // Wenn findUserByEmail aufgerufen wird, wird false zurückgegeben, um
    // anzuzeigen, dass die E-Mail nicht dupliziert ist.
    when(userDAO.findUserByEmail(user, null)).thenReturn(false);

    // Ausführung der zu testenden Methode
    String outcome = registrationBacking.register();

    // Überprüfung des erwarteten Verhaltens:
    // Der UserDAO-Mock sollte einmal die add-Methode mit dem Benutzer
    //und null als Transaction-Parameter aufgerufen haben.

    verify(userDAO, times(1)).add(user, null);

    // Das erwartete Ergebnis sollte "login" sein, da die Registrierung
    // erfolgreich war.
    assertEquals("login", outcome);
}

@Test
void testRegister_DuplicateEmail() {
    // Erzeugung eines Testbenutzers
    User user = new User();
    user.setEmail("test@example.com");

    // Verhalten des UserDAO-Mocks festlegen:
    // Wenn findUserByEmail aufgerufen wird, wird true zurückgegeben,
    //um anzuzeigen, dass die E-Mail bereits existiert.
```

```
        when(userDAO.findUserByEmail(user, null)).thenReturn(true);

        // Ausführung der zu testenden Methode
        String outcome = registrationBacking.register();

        // Überprüfung des erwarteten Verhaltens:
        // Der UserDAO-Mock sollte niemals die add-Methode aufrufen, da
        // die E-Mail bereits vorhanden ist.

        verify(userDAO, never()).add(user, null);

        // Der FacesContext-Mock sollte einmal die addMessage-Methode aufrufen,
        // um eine Fehlermeldung hinzuzufügen.
        verify(facesContext, times(1)).addMessage(null, new
         FacesMessage(FacesMessage.SEVERITY_ERROR, "Email already exists", null));

        // Das erwartete Ergebnis sollte null sein, da die Registrierung
        // fehlgeschlagen ist.
        assertEquals(null, outcome);
    }

}
```