Sure, I'll convert each of your Java files (.java) into plain text files (.txt) and provide them to you.

1. DisbursementServiceTest.java -> DisbursementServiceTest.txt

Plaintext

```
package com.professionalloan.management;

import com.professionalloan.management.model.Disbursement;
import com.professionalloan.management.model.LoanApplication;
import com.professionalloan.management.model.User;
import com.professionalloan.management.model.DisbursementStatus;
import com.professionalloan.management.model.ApplicationStatus;
import com.professionalloan.management.repository.DisbursementRepository;
import com.professionalloan.management.repository.LoanApplicationRepository;
import com.professionalloan.management.service.DisbursementService;
import com.professionalloan.management.service.NotificationService;
import com.professionalloan.management.service.RepaymentService;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;

import java.math.BigDecimal;
import java.time.LocalDate;
import java.util.Optional;

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

public class DisbursementServiceTest {

    @InjectMocks
    private DisbursementService disbursementService;
```

```
    @Mock
    private DisbursementRepository disbursementRepo;

    @Mock
    private LoanApplicationRepository loanRepo;

    @Mock
    private NotificationService notificationService;

    @Mock
    private RepaymentService repaymentService;

    private LoanApplication testLoanApplication;
    private User testUser;
    private Disbursement testDisbursement;

    @BeforeEach
    void setUp() {
        MockitoAnnotations.openMocks(this);

        testUser = new User();
        testUser.setId(1L);
        testUser.setEmail("test@example.com");

        testLoanApplication = new LoanApplication();
        testLoanApplication.setApplicationId("LOAN123");
        testLoanApplication.setUser(testUser);
        testLoanApplication.setLoanAmount(BigDecimal.valueOf(100000));
        testLoanApplication.setStatus(ApplicationStatus.APPROVED);

        testDisbursement = new Disbursement();
        testDisbursement.setId(1L);
        testDisbursement.setLoanApplication(testLoanApplication);
        testDisbursement.setAmount(BigDecimal.valueOf(100000));
        testDisbursement.setStatus(DisbursementStatus.PENDING);
        testDisbursement.setDisbursementDate(LocalDate.now());
    }
```

```
    @Test
    void testInitiateDisbursement() {
        // Arrange

when(loanRepo.findById("LOAN123")).thenReturn(Optional.of(testLoanApplication));

when(disbursementRepo.save(any(Disbursement.class))).thenReturn(testDisburseme
nt);
        doNothing().when(notificationService).notifyDisbursement(anyLong(),
anyString());
        when(repaymentService.generateEMISchedule(anyString(),
anyInt())).thenReturn(null);

        // Act
        Disbursement result = disbursementService.initiateDisbursement("LOAN123", 12);

        // Assert
        assertNotNull(result);
        assertEquals(testLoanApplication, result.getLoanApplication());
        assertEquals(BigDecimal.valueOf(100000), result.getAmount());
        assertEquals(DisbursementStatus.PENDING, result.getStatus());
        verify(disbursementRepo, times(1)).save(any(Disbursement.class));
        verify(notificationService, times(1)).notifyDisbursement(anyLong(), anyString());
        verify(repaymentService, times(1)).generateEMISchedule(anyString(), anyInt());
    }

    @Test
    void testInitiateDisbursementWithInvalidLoan() {
        // Arrange
        when(loanRepo.findById("INVALID123")).thenReturn(Optional.empty());

        // Act & Assert
        assertThrows(RuntimeException.class, () ->
            disbursementService.initiateDisbursement("INVALID123", 12)
        );
    }

    @Test
    void testInitiateDisbursementWithNonApprovedLoan() {
```

```
        // Arrange
        testLoanApplication.setStatus(ApplicationStatus.PENDING);

when(loanRepo.findById("LOAN123")).thenReturn(Optional.of(testLoanApplication));

        // Act & Assert
        assertThrows(RuntimeException.class, () ->
            disbursementService.initiateDisbursement("LOAN123", 12)
        );
    }

    @Test
    void testUpdateDisbursementStatus() {
        // Arrange
        testDisbursement.setStatus(DisbursementStatus.PENDING);

when(disbursementRepo.findById(1L)).thenReturn(Optional.of(testDisbursement));

when(disbursementRepo.save(any(Disbursement.class))).thenReturn(testDisburseme
nt);

        // Act
        Disbursement result = disbursementService.updateDisbursementStatus(1L,
DisbursementStatus.COMPLETED);

        // Assert
        assertNotNull(result);
        assertEquals(DisbursementStatus.COMPLETED, result.getStatus());
        verify(disbursementRepo, times(1)).save(testDisbursement);
    }

    @Test
    void testUpdateDisbursementStatusWithInvalidId() {
        // Arrange
        when(disbursementRepo.findById(999L)).thenReturn(Optional.empty());

        // Act & Assert
        assertThrows(RuntimeException.class, () ->
            disbursementService.updateDisbursementStatus(999L,
```

```
DisbursementStatus.COMPLETED)
        );
    }

    @Test
    void testGetDisbursementById() {
        // Arrange

when(disbursementRepo.findById(1L)).thenReturn(Optional.of(testDisbursement));

        // Act
        Disbursement result = disbursementService.getDisbursementById(1L);

        // Assert
        assertNotNull(result);
        assertEquals(testDisbursement.getId(), result.getId());
        assertEquals(testDisbursement.getAmount(), result.getAmount());
    }

    @Test
    void testGetDisbursementByIdNotFound() {
        // Arrange
        when(disbursementRepo.findById(999L)).thenReturn(Optional.empty());

        // Act
        Disbursement result = disbursementService.getDisbursementById(999L);

        // Assert
        assertNull(result);
    }
}
```

2. DocumentServiceTest.java -> DocumentServiceTest.txt

```
Plaintext
```

```
package com.professionalloan.management;

import com.professionalloan.management.model.Document;
import com.professionalloan.management.model.User;
import com.professionalloan.management.repository.DocumentRepository;
import com.professionalloan.management.repository.UserRepository;
import com.professionalloan.management.service.DocumentService;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;
import org.springframework.core.io.Resource;
import org.springframework.mock.web.MockMultipartFile;
import org.springframework.web.multipart.MultipartFile;

import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.Arrays;
import java.util.List;
import java.util.Optional;

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

public class DocumentServiceTest {

    @InjectMocks
    private DocumentService documentService;

    @Mock
    private DocumentRepository documentRepository;

    @Mock
    private UserRepository userRepository;

    private User testUser;
```

```
    private Document testDocument;
    private MultipartFile testFile;

    @BeforeEach
    void setUp() {
        MockitoAnnotations.openMocks(this);

        // Setup test user
        testUser = new User();
        testUser.setId(1L);
        testUser.setEmail("test@example.com");

        // Setup test document
        testDocument = new Document();
        testDocument.setId(1L);
        testDocument.setUser(testUser);
        testDocument.setFileType("PDF");
        testDocument.setFilePath("/test/path/document.pdf");
        testDocument.setVerified(false);

        // Setup test file
        testFile = new MockMultipartFile(
            "test.pdf",
            "test.pdf",
            "application/pdf",
            "test content".getBytes()
        );
    }

    @Test
    void testSaveDocument() {
        // Arrange
        when(userRepository.findById(1L)).thenReturn(Optional.of(testUser));

when(documentRepository.save(any(Document.class))).thenReturn(testDocument);

        // Act
        Document savedDocument = documentService.saveDocument(testFile, 1L,
"PDF");
```

```
        // Assert
        assertNotNull(savedDocument);
        assertEquals(testDocument.getFileType(), savedDocument.getFileType());
        verify(documentRepository, times(1)).save(any(Document.class));
    }

    @Test
    void testGetUserDocuments() {
        // Arrange
        List<Document> documents = Arrays.asList(testDocument);
        when(documentRepository.findByUser_Id(1L)).thenReturn(documents);

        // Act
        List<Document> result = documentService.getUserDocuments(1L);

        // Assert
        assertNotNull(result);
        assertEquals(1, result.size());
        assertEquals(testDocument.getId(), result.get(0).getId());
    }

    @Test
    void testDeleteDocument() {
        // Arrange
        when(documentRepository.findById(1L)).thenReturn(Optional.of(testDocument));
        doNothing().when(documentRepository).delete(testDocument);

        // Act & Assert
        assertDoesNotThrow(() -> documentService.deleteDocument(1L));
        verify(documentRepository, times(1)).delete(testDocument);
    }

    @Test
    void testVerifyDocument() {
        // Arrange
        when(documentRepository.findById(1L)).thenReturn(Optional.of(testDocument));

when(documentRepository.save(any(Document.class))).thenReturn(testDocument);
```

```
        // Act
        documentService.verifyDocument(1L);

        // Assert
        assertTrue(testDocument.isVerified());
        verify(documentRepository, times(1)).save(testDocument);
    }

    @Test
    void testGetUserDocumentsByType() {
        // Arrange
        List<Document> documents = Arrays.asList(testDocument);
        when(documentRepository.findByUser_IdAndFileType(1L,
"PDF")).thenReturn(documents);

        // Act
        List<Document> result = documentService.getUserDocumentsByType(1L, "PDF");

        // Assert
        assertNotNull(result);
        assertEquals(1, result.size());
        assertEquals("PDF", result.get(0).getFileType());
    }

    @Test
    void testSaveDocumentWithInvalidUser() {
        // Arrange
        when(userRepository.findById(999L)).thenReturn(Optional.empty());

        // Act & Assert
        assertThrows(RuntimeException.class, () ->
            documentService.saveDocument(testFile, 999L, "PDF")
        );
    }

    @Test
    void testSaveDocumentWithNullFile() {
        // Act & Assert
```

```
        assertThrows(IllegalArgumentException.class, () ->
            documentService.saveDocument(null, 1L, "PDF")
        );
    }
}
```

3. EmailServiceTest.java -> EmailServiceTest.txt

Plaintext

```
package com.professionalloan.management;

import com.professionalloan.management.service.EmailService;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;
import org.springframework.mail.SimpleMailMessage;
import org.springframework.mail.javamail.JavaMailSender;

import static org.mockito.ArgumentMatchers.any;
import static org.mockito.Mockito.*;

public class EmailServiceTest {

    @InjectMocks
    private EmailService emailService;

    @Mock
    private JavaMailSender mailSender;

    @BeforeEach
    void setUp() {
        MockitoAnnotations.openMocks(this);
    }
```

```
    @Test
    void testSendOtpEmail() {
        // Arrange
        String testEmail = "test@example.com";
        String testOtp = "123456";
        doNothing().when(mailSender).send(any(SimpleMailMessage.class));

        // Act
        emailService.sendOtpEmail(testEmail, testOtp);

        // Assert
        verify(mailSender, times(1)).send(any(SimpleMailMessage.class));
    }

    @Test
    void testSendOtpEmailVerifyContent() {
        // Arrange
        String testEmail = "test@example.com";
        String testOtp = "123456";

        // Create a message capture
        SimpleMailMessage capturedMessage = new SimpleMailMessage();
        doAnswer(invocation -> {
            SimpleMailMessage message = invocation.getArgument(0);
            capturedMessage.setTo(message.getTo());
            capturedMessage.setSubject(message.getSubject());
            capturedMessage.setText(message.getText());
            capturedMessage.setFrom(message.getFrom());
            return null;
        }).when(mailSender).send(any(SimpleMailMessage.class));

        // Act
        emailService.sendOtpEmail(testEmail, testOtp);

        // Assert
        verify(mailSender, times(1)).send(any(SimpleMailMessage.class));
        assert capturedMessage.getTo()[0].equals(testEmail);
        assert capturedMessage.getSubject().equals("Your OTP for Password Reset");
```

```
        assert capturedMessage.getText().contains(testOtp);
        assert capturedMessage.getFrom().equals("avijit.dam9@gmail.com");
    }

    @Test
    void testSendOtpEmailWithNullValues() {
        // Act & Assert
        emailService.sendOtpEmail(null, null);
        verify(mailSender, times(1)).send(any(SimpleMailMessage.class));
    }
}
```

4. LoanApplicationServiceTest.java -> LoanApplicationServiceTest.txt

Plaintext

```
package com.professionalloan.management;
import com.professionalloan.management.service.LoanApplicationService;
import com.professionalloan.management.service.NotificationService;
import com.professionalloan.management.service.DocumentService;

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

import java.math.BigDecimal;
import java.util.Optional;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;

import com.professionalloan.management.model.LoanApplication;
import com.professionalloan.management.model.ApplicationStatus;
import com.professionalloan.management.repository.LoanApplicationRepository;
```

```
import com.professionalloan.management.repository.UserRepository;


public class LoanApplicationServiceTest {

    @InjectMocks
    private LoanApplicationService loanApplicationService;

    @Mock
    private LoanApplicationRepository loanApplicationRepository;

    @Mock
    private UserRepository userRepository;

    @Mock
    private NotificationService notificationService;

    @Mock
    private DocumentService documentService;

    @BeforeEach
    void setUp() {
        MockitoAnnotations.openMocks(this);
    }

    @Test
    void testGetApplicationById() {
        LoanApplication loanApplication = new LoanApplication();
        loanApplication.setApplicationId("APP123");

when(loanApplicationRepository.findById("APP123")).thenReturn(Optional.of(loanAppli
cation));

        LoanApplication found = loanApplicationService.getApplicationById("APP123");
        assertNotNull(found);
        assertEquals("APP123", found.getApplicationId());
    }

    @Test
```

```
    void testUpdateLoanStatus() {
        LoanApplication loanApplication = new LoanApplication();
        loanApplication.setApplicationId("APP456");
        loanApplication.setStatus(ApplicationStatus.PENDING);


when(loanApplicationRepository.findById("APP456")).thenReturn(Optional.of(loanAppl
ication));

when(loanApplicationRepository.save(any(LoanApplication.class))).thenReturn(loanAp
plication);

        LoanApplication updated = loanApplicationService.updateLoanStatus("APP456",
ApplicationStatus.APPROVED);
        assertEquals(ApplicationStatus.APPROVED, updated.getStatus());
    }
}
```

5. LoanManagementSystemAlpa2ApplicationTests.java -> LoanManagementSystemAlpa2ApplicationTests.txt

Plaintext

```
package com.professionalloan.management;

import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
class LoanManagementSystemAlpa2ApplicationTests {

        @Test
        void contextLoads() {
        }

}
```

6. NotificationServiceTest.java -> NotificationServiceTest.txt

Plaintext

```
package com.professionalloan.management;

import com.professionalloan.management.model.Notification;
import com.professionalloan.management.model.User;
import com.professionalloan.management.model.ApplicationStatus;
import com.professionalloan.management.repository.NotificationRepository;
import com.professionalloan.management.repository.UserRepository;
import com.professionalloan.management.service.NotificationService;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;
import org.springframework.mail.SimpleMailMessage;
import org.springframework.mail.javamail.JavaMailSender;

import java.util.Arrays;
import java.util.List;
import java.util.Optional;

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

public class NotificationServiceTest {

    @InjectMocks
    private NotificationService notificationService;

    @Mock
    private NotificationRepository notificationRepository;

    @Mock
    private UserRepository userRepository;

    @Mock
    private JavaMailSender mailSender;

    private User testUser;
    private Notification testNotification;
```

```
    @BeforeEach
    void setUp() {
        MockitoAnnotations.openMocks(this);

        testUser = new User();
        testUser.setId(1L);
        testUser.setEmail("test@example.com");

        testNotification = new Notification();
        testNotification.setId(1L);
        testNotification.setUser(testUser);
        testNotification.setMessage("Test notification");
        testNotification.setType("TEST");
        testNotification.setRead(false);
    }

    @Test
    void testCreateNotification() {
        // Arrange
        when(userRepository.findById(1L)).thenReturn(Optional.of(testUser));
        when(notificationRepository.save(any(Notification.class))).thenReturn(testNotification);

        // Act
        Notification result = notificationService.createNotification(1L, "Test notification", "TEST");

        // Assert
        assertNotNull(result);
        assertEquals(testUser, result.getUser());
        assertEquals("Test notification", result.getMessage());
        assertEquals("TEST", result.getType());
        assertFalse(result.isRead());
        verify(notificationRepository, times(1)).save(any(Notification.class));
    }

    @Test
    void testCreateNotificationWithInvalidUser() {
        // Arrange
        when(userRepository.findById(999L)).thenReturn(Optional.empty());

        // Act & Assert
        assertThrows(RuntimeException.class, () ->
            notificationService.createNotification(999L, "Test notification", "TEST")
        );
    }
```

```
    @Test
    void testGetUserNotifications() {
        // Arrange
        List<Notification> notifications = Arrays.asList(testNotification);

when(notificationRepository.findByUser_IdOrderByCreatedAtDesc(1L)).thenReturn(notifications);

        // Act
        List<Notification> result = notificationService.getUserNotifications(1L);

        // Assert
        assertNotNull(result);
        assertEquals(1, result.size());
        assertEquals(testNotification.getMessage(), result.get(0).getMessage());
    }

    @Test
    void testMarkAsRead() {
        // Arrange
        when(notificationRepository.findById(1L)).thenReturn(Optional.of(testNotification));
        when(notificationRepository.save(any(Notification.class))).thenReturn(testNotification);

        // Act
        Notification result = notificationService.markAsRead(1L);

        // Assert
        assertTrue(result.isRead());
        verify(notificationRepository, times(1)).save(testNotification);
    }

    @Test
    void testMarkAsReadWithInvalidId() {
        // Arrange
        when(notificationRepository.findById(999L)).thenReturn(Optional.empty());

        // Act & Assert
        assertThrows(RuntimeException.class, () ->
            notificationService.markAsRead(999L)
        );
    }

    @Test
    void testNotifyLoanStatus() {
        // Arrange
```

```
        when(userRepository.findById(1L)).thenReturn(Optional.of(testUser));
        when(notificationRepository.save(any(Notification.class))).thenReturn(testNotification);

        // Act
        notificationService.notifyLoanStatus(1L, "LOAN123", ApplicationStatus.APPROVED);

        // Assert
        verify(notificationRepository, times(1)).save(any(Notification.class));
    }

    @Test
    void testNotifyEMIDue() {
        // Arrange
        when(userRepository.findById(1L)).thenReturn(Optional.of(testUser));
        when(notificationRepository.save(any(Notification.class))).thenReturn(testNotification);

        // Act
        notificationService.notifyEMIDue(1L, "LOAN123", 1);

        // Assert
        verify(notificationRepository, times(1)).save(any(Notification.class));
    }

    @Test
    void testNotifyEMIOverdue() {
        // Arrange
        when(userRepository.findById(1L)).thenReturn(Optional.of(testUser));
        when(notificationRepository.save(any(Notification.class))).thenReturn(testNotification);

        // Act
        notificationService.notifyEMIOverdue(1L, "LOAN123", 1);

        // Assert
        verify(notificationRepository, times(1)).save(any(Notification.class));
    }

    @Test
    void testNotifyDisbursement() {
        // Arrange
        when(userRepository.findById(1L)).thenReturn(Optional.of(testUser));
        when(notificationRepository.save(any(Notification.class))).thenReturn(testNotification);

        // Act
        notificationService.notifyDisbursement(1L, "LOAN123");
```

```
        // Assert
        verify(notificationRepository, times(1)).save(any(Notification.class));
    }
}
```

7. RepaymentServiceTest.java -> RepaymentServiceTest.txt

Plaintext

```
package com.professionalloan.management;

import com.professionalloan.management.model.LoanApplication;
import com.professionalloan.management.model.Repayment;
import com.professionalloan.management.repository.LoanApplicationRepository;
import com.professionalloan.management.repository.RepaymentRepository;
import com.professionalloan.management.service.RepaymentService;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;

import java.math.BigDecimal;
import java.time.LocalDate;
import java.util.List;
import java.util.Optional;

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

public class RepaymentServiceTest {

    @InjectMocks
    private RepaymentService repaymentService;

    @Mock
    private RepaymentRepository repaymentRepository;

    @Mock
    private LoanApplicationRepository loanApplicationRepository;

    private LoanApplication testLoanApplication;
```

```
    @BeforeEach
    void setUp() {
        MockitoAnnotations.openMocks(this);

        testLoanApplication = new LoanApplication();
        testLoanApplication.setApplicationId("TEST123");
        testLoanApplication.setLoanAmount(BigDecimal.valueOf(100000));
    }

    @Test
    void testCalculateEMI() {
        // Test case for 100000 principal, 12 months tenure, 12% interest
        BigDecimal principal = BigDecimal.valueOf(100000);
        int tenureInMonths = 12;
        double interestRate = 12.0;

        BigDecimal emi = repaymentService.calculateEMI(principal, tenureInMonths, interestRate);

        assertNotNull(emi);
        assertTrue(emi.compareTo(BigDecimal.ZERO) > 0);
    }

    @Test
    void testGenerateEMISchedule() {
        // Arrange

when(loanApplicationRepository.findById("TEST123")).thenReturn(Optional.of(testLoanApplication
));
        when(repaymentRepository.saveAll(any())).thenAnswer(invocation ->
invocation.getArgument(0));

        // Act
        List<Repayment> schedule = repaymentService.generateEMISchedule("TEST123", 12);

        // Assert
        assertNotNull(schedule);
        assertEquals(12, schedule.size());

        // Verify first EMI
        Repayment firstEMI = schedule.get(0);
        assertEquals(1, firstEMI.getEmiNumber());
        assertEquals("PENDING", firstEMI.getStatus());
        assertEquals(testLoanApplication, firstEMI.getLoanApplication());
        assertNotNull(firstEMI.getDueDate());
```

```
        // Verify last EMI
        Repayment lastEMI = schedule.get(11);
        assertEquals(12, lastEMI.getEmiNumber());
        assertEquals("PENDING", lastEMI.getStatus());
        assertEquals(testLoanApplication, lastEMI.getLoanApplication());
    }

    @Test
    void testGenerateEMIScheduleWithInvalidLoan() {
        // Arrange
        when(loanApplicationRepository.findById("INVALID123")).thenReturn(Optional.empty());

        // Act & Assert
        assertThrows(RuntimeException.class, () ->
            repaymentService.generateEMISchedule("INVALID123", 12)
        );
    }

    @Test
    void testCalculateEMIWithZeroPrincipal() {
        // Arrange
        BigDecimal principal = BigDecimal.ZERO;
        int tenureInMonths = 12;
        double interestRate = 12.0;

        // Act
        BigDecimal emi = repaymentService.calculateEMI(principal, tenureInMonths, interestRate);

        // Assert
        assertEquals(BigDecimal.ZERO.setScale(2), emi);
    }

    @Test
    void testCalculateEMIWithZeroInterest() {
        // Arrange
        BigDecimal principal = BigDecimal.valueOf(12000);
        int tenureInMonths = 12;
        double interestRate = 0.0;

        // Act
        BigDecimal emi = repaymentService.calculateEMI(principal, tenureInMonths, interestRate);

        // Assert
        assertEquals(BigDecimal.valueOf(1000).setScale(2), emi);
    }
```

```
}
```

8. UserServiceTest.java -> UserServiceTest.txt

```
Plaintext

package com.professionalloan.management;

import com.professionalloan.management.model.User;
import com.professionalloan.management.repository.UserRepository;
import com.professionalloan.management.service.UserService;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;

import java.util.Optional;

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

public class UserServiceTest {

    @InjectMocks
    private UserService userService;

    @Mock
    private UserRepository userRepository;

    private User testUser;

    @BeforeEach
    void setUp() {
        MockitoAnnotations.openMocks(this);

        testUser = new User();
        testUser.setId(1L);
        testUser.setEmail("test@example.com");
        testUser.setPassword("password123");
    }

    @Test
```

```
    void testRegisterUserSuccess() {
        // Arrange
        when(userRepository.findByEmail(testUser.getEmail())).thenReturn(Optional.empty());
        when(userRepository.save(any(User.class))).thenReturn(testUser);

        // Act
        String result = userService.registerUser(testUser);

        // Assert
        assertEquals("Registration successful!", result);
        verify(userRepository, times(1)).save(testUser);
    }

    @Test
    void testRegisterUserWithExistingEmail() {
        // Arrange
        when(userRepository.findByEmail(testUser.getEmail())).thenReturn(Optional.of(testUser));

        // Act
        String result = userService.registerUser(testUser);

        // Assert
        assertEquals("Email already registered!", result);
        verify(userRepository, never()).save(any(User.class));
    }

    @Test
    void testFindByEmailAndPasswordSuccess() {
        // Arrange
        when(userRepository.findByEmail(testUser.getEmail())).thenReturn(Optional.of(testUser));

        // Act
        User result = userService.findByEmailAndPassword(testUser.getEmail(),
testUser.getPassword());

        // Assert
        assertNotNull(result);
        assertEquals(testUser.getEmail(), result.getEmail());
        assertEquals(testUser.getPassword(), result.getPassword());
    }

    @Test
    void testFindByEmailAndPasswordWithWrongPassword() {
        // Arrange
        when(userRepository.findByEmail(testUser.getEmail())).thenReturn(Optional.of(testUser));
```

```
        // Act
        User result = userService.findByEmailAndPassword(testUser.getEmail(), "wrongpassword");

        // Assert
        assertNull(result);
    }

    @Test
    void testFindByEmailAndPasswordWithNonexistentEmail() {
        // Arrange

when(userRepository.findByEmail("nonexistent@example.com")).thenReturn(Optional.empty());

        // Act
        User result = userService.findByEmailAndPassword("nonexistent@example.com",
"password123");

        // Assert
        assertNull(result);
    }

    @Test
    void testFindByEmailSuccess() {
        // Arrange
        when(userRepository.findByEmail(testUser.getEmail())).thenReturn(Optional.of(testUser));

        // Act
        User result = userService.findByEmail(testUser.getEmail());

        // Assert
        assertNotNull(result);
        assertEquals(testUser.getEmail(), result.getEmail());
    }

    @Test
    void testFindByEmailNotFound() {
        // Arrange

when(userRepository.findByEmail("nonexistent@example.com")).thenReturn(Optional.empty());

        // Act
        User result = userService.findByEmail("nonexistent@example.com");

        // Assert
```

```
        assertNull(result);
    }

    @Test
    void testSaveUser() {
        // Arrange
        when(userRepository.save(any(User.class))).thenReturn(testUser);

        // Act
        userService.save(testUser);

        // Assert
        verify(userRepository, times(1)).save(testUser);
    }
}
```