

**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА**

Факультет комп'ютерних наук та кібернетики

Звіт

Валідація та верифікація програмних систем

Mock tests

Виконав студент

групи ІНФ-1

Веремчук Максим Романович

Опис інструментарію

В ході виконання лабораторної роботи проводилось тестування примітивної банківської системи. Використовувалась мова програмування java в редакторі intellij Idea на системі Ubuntu 18.04. Використовувались допоміжні бібліотеки Java Spring та Mockito для тестування.

Порядок роботи

Система, яку потрібно протестувати, являє собою систему електронних платежів для внесення/зняття готівки, блокування/видача картки та інші стандартні банківські операції. При реалізації створені dao класи bank, payment, card, account. Для них створені відповідні mock, за допомогою ініціалізації:

```
@Before
public void init(){
    MockitoAnnotations.initMocks(this);
    mockMvc = MockMvcBuilders
        .standaloneSetup(userController)
        .build();

    ReflectionTestUtils.setField(businessService, "accountDao",
mock(AccountDao.class));
    ReflectionTestUtils.setField(businessService, "cardDao",
mock(CardDao.class));
    ReflectionTestUtils.setField(businessService, "paymentDao",
mock(PaymentDao.class));
    ReflectionTestUtils.setField(businessService, "bankDao",
mock(BankDao.class));
}
```

Для тестування використовуються mock об'єкти класу аутентифікації:

```
Authentication authentication = mock(Authentication.class);
SecurityContext securityContext = mock(SecurityContext.class);
```

```

when(authentication.getPrincipal()).thenReturn("xyz");
when(securityContext.getAuthentication()).thenReturn(authentication);
SecurityContextHolder.setContext(securityContext);

```

Аналогічні тести оплати та верифікація юзера:

```

public void setSecurityContext(String role){
    UserProfile userProfile = new UserProfile();
    userProfile.setType(role);
    User user = new User();
    user.setUsername("xyz");
    Set<UserProfile> userProfileSet = new HashSet<>();
    userProfileSet.add(userProfile);
    user.setUserProfiles(userProfileSet);
    when(userService.findByUsername("xyz")).thenReturn(user);

when(userProfileService.findByType("User")).thenReturn(userProfile);
    Authentication authentication = mock(Authentication.class);
    SecurityContext securityContext = mock(SecurityContext.class);
    when(authentication.getPrincipal()).thenReturn("xyz");

when(securityContext.getAuthentication()).thenReturn(authentication);
    SecurityContextHolder.setContext(securityContext);

}

@Test
public void submitRegistration() throws Exception {
    when(userService.isUserNameUnique("xyz")).thenReturn(true);
    when(userProfileService.findByType("User")).thenReturn(new
UserProfile());
    this.mockMvc
        .perform(
            post("/newuser")

.contentType(MediaType.APPLICATION_FORM_URLENCODED)
                .param("name", "xyz")
                .param("username", "xyz")
                .param("password", "xyz")
            )
        .andExpect(status().isOk());
}

@Test
public void submitRegistrationWithWrongName() throws Exception {
    when(userService.isUserNameUnique("xyz")).thenReturn(false);

```

```

        this.mockMvc
            .perform(
                post("/newuser")

.contentType(MediaType.APPLICATION_FORM_URLENCODED)
                .param("name", "xyz")
                .param("username", "xyz")
                .param("password", "xyz")
            )
            .andExpect(status().isOk());
    }

    @Test
    public void loginUser() throws Exception {
        setSecurityContext("USER");

        this.mockMvc
            .perform(
                get("/login")

.contentType(MediaType.APPLICATION_FORM_URLENCODED)
                .param("username", "xyz")
                .param("password", "xyz")
            )
            .andExpect(redirectedUrl("/userPage"))
            .andExpect(status().is3xxRedirection());
    }

    @Test
    public void loginAdmin() throws Exception {
        setSecurityContext("ADMIN");
        this.mockMvc
            .perform(
                get("/login")

.contentType(MediaType.APPLICATION_FORM_URLENCODED)
                .param("username", "xyz")
                .param("password", "xyz")
            )
            .andExpect(redirectedUrl("/adminPage"))
            .andExpect(status().is3xxRedirection());
    }

    @Test
    public void PayBlockedCard() throws Exception {

```

```

        setSecurityContext("USER");
        Map<Card, Account> cardAccountMap = new HashMap<>();
        Card card = new Card();
        card.setCardNumber(123);
        Account account = new Account();
        account.setMoneyAmount(10);
        cardAccountMap.put(card, account);
        MockHttpSession session = mock(MockHttpSession.class);

when(session.getAttribute("cardAccountMap")).thenReturn(cardAccountMap)
;

        this.mockMvc
            .perform(
                post("/operation")
                    .session(session)

.contentType(MediaType.APPLICATION_FORM_URLENCODED)
                    .param("card", "123")
                    .param("command", "Block")
            )
            .andExpect(model().attribute("warning", "Warning:
blocked"))
            .andExpect(status().isOk());
        this.mockMvc
            .perform(
                post("/operation")
                    .session(session)

.contentType(MediaType.APPLICATION_FORM_URLENCODED)
                    .param("money", "1")
                    .param("info", "info")
                    .param("card", "123")
                    .param("command", "Pay")
            )
            .andExpect(model().attribute("warning", "Warning:
Account is blocked"))
            .andExpect(status().isOk());
    }

    @Test
    public void PaySuccess() throws Exception {
        setSecurityContext("USER");
        Map<Card, Account> cardAccountMap = new HashMap<>();
        Card card = new Card();

```

```

        card.setCardNumber(123);
        Account account = new Account();
        account.setMoneyAmount(10);
        cardAccountMap.put(card, account);
        MockHttpSession session = mock(MockHttpSession.class);

when(session.getAttribute("cardAccountMap")).thenReturn(cardAccountMap)
;

        User user = new User();
        user.setUsername("xyz");
        user.setId(1);
        when(session.getAttribute("User")).thenReturn(user);

when(session.getAttribute("cardAccountMap")).thenReturn(cardAccountMap)
;

        this.mockMvc
                .perform(
                        post("/operation")
                                .session(session)

.contentType(MediaType.APPLICATION_FORM_URLENCODED)
                                .param("money", "1")
                                .param("info", "info")
                                .param("card", "123")
                                .param("command", "Pay")
                        )
                .andExpect(model().attribute("testPayment", "info 1 9
123"))
                .andExpect(status().isOk());
    }

    @Test
    public void TopUpBlockedCard() throws Exception {
        setSecurityContext("USER");
        Map<Card, Account> cardAccountMap = new HashMap<>();
        Card card = new Card();
        card.setCardNumber(123);
        Account account = new Account();
        account.setMoneyAmount(10);
        cardAccountMap.put(card, account);
        MockHttpSession session = mock(MockHttpSession.class);

when(session.getAttribute("cardAccountMap")).thenReturn(cardAccountMap)
;

```

```

        this.mockMvc
            .perform(
                post("/operation")
                    .session(session)

.contentType(MediaType.APPLICATION_FORM_URLENCODED)
                    .param("card", "123")
                    .param("command", "Block")
            )
            .andExpect(model().attribute("warning", "Warning:
blocked"))
            .andExpect(status().isOk());
        this.mockMvc
            .perform(
                post("/operation")
                    .session(session)

.contentType(MediaType.APPLICATION_FORM_URLENCODED)
                    .param("money", "1")
                    .param("info", "info")
                    .param("card", "123")
                    .param("command", "TopUp")
            )
            .andExpect(model().attribute("warning", "Warning:
Account is blocked"))
            .andExpect(status().isOk());
    }

    @Test
    public void TopUpSuccess() throws Exception {
        setSecurityContext("USER");
        Map<Card, Account> cardAccountMap = new HashMap<>();
        Card card = new Card();
        card.setCardNumber(123);
        Account account = new Account();
        account.setMoneyAmount(10);
        cardAccountMap.put(card, account);
        MockHttpSession session = mock(MockHttpSession.class);

when(session.getAttribute("cardAccountMap")).thenReturn(cardAccountMap)
;

        User user = new User();
        user.setUsername("xyz");
        user.setId(1);
    }

```

```

when(session.getAttribute("User")).thenReturn(user);
this.mockMvc
    .perform(
        post("/operation")
            .session(session)

.contentType(MediaType.APPLICATION_FORM_URLENCODED)
            .param("money", "1")
            .param("info", "info")
            .param("card", "123")
            .param("command", "TopUp")
        )
    .andExpect(model().attribute("topup", "+1$ to
account."))
    .andExpect(status().isOk());

}
}

```

Висновки

У ході виконання лабораторної роботи було виконано mock об'єкти для тестування програми банківської системи.