tactusoft / loteriabogota

Code Issues Pull requests Actions Projects Security and quality Insi

We are having a problem billing the locqum organization. Please update your payment method or call your payment provider for details on why the transaction failed.
You can always contact support with any questions.

Commit 6b1856d

claude committed last month Verified

```
test: agregar pruebas unitarias para servicios de dominio y VOs

Tests unitarios con JUnit 5 + Mockito para:
- LotteryService (4 tests): listado activos, busqueda por id, not found
- TicketService (9 tests): compra valida, validaciones de numero/serie/fracciones,
  checkout con usuario nuevo/existente, items vacios
- PaymentService (8 tests): creacion con email/phone, validaciones,
  check status aprobado/rechazado/ya procesado, orden no encontrada
- AuthService (7 tests): registro, login, credenciales invalidas, refresh token
- PhoneVerificationService (9 tests): envio OTP, verificacion valida/invalida,
  expirado, ya usado, demasiados intentos, isVerified
- UserService (4 tests): busqueda, actualizacion parcial/total de perfil
- ValueObjects (15 tests): Email, PhoneNumber, Money, DocumentId con validaciones

Total: 56 tests unitarios cubriendo la capa de dominio.

https://claude.ai/code/session_01WTbEJHa9nd8kVVoKCirXH8
```

master

1 parent 24c920c commit 6b1856d

7 files changed +984 -0 lines changed

Top

Filter files...

- backend/src/test/java/co/gov/loteriabogota/loteriabackend/domain
  - service
    - AuthServiceTest.java
    - LotteryServiceTest.java
    - PaymentServiceTest.java

PhoneVerificationServiceTest.java

TicketServiceTest.java

UserServiceTest.java

vo

ValueObjectsTest.java

7 files changed +984 -0 lines changed

Search within code

…iabackend/domain/service/AuthServiceTest.java

```
@@ -0,0 +1,127 @@
+ package co.gov.loteriabogota.loteriabackend.domain.service;
+
+ import co.gov.loteriabogota.loteriabackend.domain.exception.DomainException;
+ import co.gov.loteriabogota.loteriabackend.domain.model.User;
+ import co.gov.loteriabogota.loteriabackend.domain.port.in.AuthUseCase;
+ import co.gov.loteriabogota.loteriabackend.domain.port.out.UserRepository;
+ import co.gov.loteriabogota.loteriabackend.domain.vo.Email;
+ import co.gov.loteriabogota.loteriabackend.infrastructure.security.JwtProvider;
+ 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.Mock;
+ import org.mockito.junit.jupiter.MockitoExtension;
+ import org.springframework.security.crypto.password.PasswordEncoder;
+
+ import java.util.Optional;
+
+ import static org.assertj.core.api.Assertions.*;
+ import static org.mockito.ArgumentMatchers.any;
+ import static org.mockito.ArgumentMatchers.anyString;
+ import static org.mockito.Mockito.*;
+
+ @ExtendWith(MockitoExtension.class)
+ class AuthServiceTest {
+
+     @Mock
+     private UserRepository userRepository;
+
```

```
30  +     @Mock
31  +     private JwtProvider jwtProvider;
32  +
33  +     @Mock
34  +     private PasswordEncoder passwordEncoder;
35  +
36  +     @InjectMocks
37  +     private AuthService authService;
38  +
39  +     private Email email;
40  +
41  +     @BeforeEach
42  +     void setUp() {
43  +         email = new Email("test@example.com");
44  +     }
45  +
46  +     @Test
47  +     void register_newUser_succeeds() {
48  +         when(userRepository.findByEmail(email)).thenReturn(Optional.empty());
49  +         when(passwordEncoder.encode("password123")).thenReturn("hashed");
50  +         when(userRepository.save(any(User.class))).thenAnswer(inv ->
      inv.getArgument(0));
51  +         when(jwtProvider.generateAccessToken(anyString())).thenReturn("access-
      token");
52  +         when(jwtProvider.generateRefreshToken(anyString())).thenReturn("refresh-
      token");
53  +
54  +         AuthUseCase.AuthResult result = authService.register(email, "password123",
      "Juan", "Perez");
55  +
56  +         assertThat(result.accessToken()).isEqualTo("access-token");
57  +         assertThat(result.refreshToken()).isEqualTo("refresh-token");
58  +         assertThat(result.newUser()).isTrue();
59  +         verify(userRepository).save(any(User.class));
60  +     }
61  +
62  +     @Test
63  +     void register_existingUser_throwsDomainException() {
64  +         User existing = User.createBasic(email);
65  +         when(userRepository.findByEmail(email)).thenReturn(Optional.of(existing));
66  +
```

```
67  +         assertThatThrownBy(() -> authService.register(email, "password123",
    "Juan", "Perez"))
68  +                 .isInstanceOf(DomainException.class)
69  +                 .hasMessageContaining("Ya existe");
70  +     }
71  +
72  +     @Test
73  +     void login_validCredentials_returnsTokens() {
74  +         User user = User.createWithPassword(email, "hashed", "Juan", "Perez");
75  +         when(userRepository.findByEmail(email)).thenReturn(Optional.of(user));
76  +         when(passwordEncoder.matches("password123", "hashed")).thenReturn(true);
77  +         when(jwtProvider.generateAccessToken(anyString())).thenReturn("access-
    token");
78  +         when(jwtProvider.generateRefreshToken(anyString())).thenReturn("refresh-
    token");
79  +
80  +         AuthUseCase.AuthResult result = authService.login(email, "password123");
81  +
82  +         assertThat(result.accessToken()).isEqualTo("access-token");
83  +         assertThat(result.newUser()).isFalse();
84  +     }
85  +
86  +     @Test
87  +     void login_wrongPassword_throwsDomainException() {
88  +         User user = User.createWithPassword(email, "hashed", "Juan", "Perez");
89  +         when(userRepository.findByEmail(email)).thenReturn(Optional.of(user));
90  +         when(passwordEncoder.matches("wrong", "hashed")).thenReturn(false);
91  +
92  +         assertThatThrownBy(() -> authService.login(email, "wrong"))
93  +                 .isInstanceOf(DomainException.class)
94  +                 .hasMessageContaining("incorrectos");
95  +     }
96  +
97  +     @Test
98  +     void login_userNotFound_throwsDomainException() {
99  +         when(userRepository.findByEmail(email)).thenReturn(Optional.empty());
100 +
101 +         assertThatThrownBy(() -> authService.login(email, "password123"))
102 +                 .isInstanceOf(DomainException.class)
103 +                 .hasMessageContaining("incorrectos");
104 +     }
105 +
```

```
106  +     @Test
107  +     void login_userWithoutPassword_throwsDomainException() {
108  +         User user = User.createBasic(email);
109  +         when(userRepository.findByEmail(email)).thenReturn(Optional.of(user));
110  +
111  +         assertThatThrownBy(() -> authService.login(email, "password123"))
112  +                 .isInstanceOf(DomainException.class)
113  +                 .hasMessageContaining("incorrectos");
114  +     }
115  +
116  +     @Test
117  +     void refreshToken_validToken_returnsNewTokens() {
118  +         when(jwtProvider.validateAndGetSubject("old-refresh")).thenReturn("user-
         id");
119  +         when(jwtProvider.generateAccessToken("user-id")).thenReturn("new-access");
120  +         when(jwtProvider.generateRefreshToken("user-id")).thenReturn("new-
         refresh");
121  +
122  +         AuthUseCase.AuthResult result = authService.refreshToken("old-refresh");
123  +
124  +         assertThat(result.accessToken()).isEqualTo("new-access");
125  +         assertThat(result.refreshToken()).isEqualTo("new-refresh");
126  +     }
127  + }
```

…ackend/domain/service/LotteryServiceTest.java

```
•••    @@ -0,0 +1,82 @@
1    + package co.gov.loteriabogota.loteriabackend.domain.service;
2    +
3    + import
       co.gov.loteriabogota.loteriabackend.domain.exception.LotteryNotFoundException;
4    + import co.gov.loteriabogota.loteriabackend.domain.model.Lottery;
5    + import co.gov.loteriabogota.loteriabackend.domain.port.out.LotteryRepository;
6    + import co.gov.loteriabogota.loteriabackend.domain.vo.Money;
7    + import org.junit.jupiter.api.BeforeEach;
8    + import org.junit.jupiter.api.Test;
9    + import org.junit.jupiter.api.extension.ExtendWith;
10   + import org.mockito.InjectMocks;
11   + import org.mockito.Mock;
12   + import org.mockito.junit.jupiter.MockitoExtension;
13   +
```

```
14  + import java.util.List;
15  + import java.util.Optional;
16  + import java.util.UUID;
17  +
18  + import static org.assertj.core.api.Assertions.*;
19  + import static org.mockito.Mockito.*;
20  +
21  + @ExtendWith(MockitoExtension.class)
22  + class LotteryServiceTest {
23  +
24  +     @Mock
25  +     private LotteryRepository lotteryRepository;
26  +
27  +     @InjectMocks
28  +     private LotteryService lotteryService;
29  +
30  +     private Lottery lottery;
31  +     private UUID lotteryId;
32  +
33  +     @BeforeEach
34  +     void setUp() {
35  +         lotteryId = UUID.randomUUID();
36  +         lottery = new Lottery();
37  +         lottery.setId(lotteryId);
38  +         lottery.setName("Loteria de Bogota");
39  +         lottery.setPricePerFraction(Money.cop(5000));
40  +         lottery.setTotalFractions(10);
41  +         lottery.setActive(true);
42  +     }
43  +
44  +     @Test
45  +     void listActiveLotteries_returnsAllActive() {
46  +         when(lotteryRepository.findAllActive()).thenReturn(List.of(lottery));
47  +
48  +         List<Lottery> result = lotteryService.listActiveLotteries();
49  +
50  +         assertThat(result).hasSize(1);
51  +         assertThat(result.get(0).getName()).isEqualTo("Loteria de Bogota");
52  +         verify(lotteryRepository).findAllActive();
53  +     }
54  +
55  +     @Test
```

```
56  +     void listActiveLotteries_emptyList() {
57  +         when(lotteryRepository.findAllActive()).thenReturn(List.of());
58  +
59  +         List<Lottery> result = lotteryService.listActiveLotteries();
60  +
61  +         assertThat(result).isEmpty();
62  +     }
63  +
64  +     @Test
65  +     void getLotteryById_found() {
66  +
        when(lotteryRepository.findById(lotteryId)).thenReturn(Optional.of(lottery));
67  +
68  +         Lottery result = lotteryService.getLotteryById(lotteryId);
69  +
70  +         assertThat(result.getId()).isEqualTo(lotteryId);
71  +         assertThat(result.getName()).isEqualTo("Loteria de Bogota");
72  +     }
73  +
74  +     @Test
75  +     void getLotteryById_notFound_throwsException() {
76  +         UUID unknownId = UUID.randomUUID();
77  +         when(lotteryRepository.findById(unknownId)).thenReturn(Optional.empty());
78  +
79  +         assertThatThrownBy(() -> lotteryService.getLotteryById(unknownId))
80  +                 .isInstanceOf(LotteryNotFoundException.class);
81  +     }
82  + }
```

…ackend/domain/service/PaymentServiceTest.java

```
    @@ -0,0 +1,210 @@
1   + package co.gov.loteriabogota.loteriabackend.domain.service;
2   +
3   + import co.gov.loteriabogota.loteriabackend.domain.exception.DomainException;
4   + import co.gov.loteriabogota.loteriabackend.domain.model.Lottery;
5   + import co.gov.loteriabogota.loteriabackend.domain.model.LotteryTicket;
6   + import co.gov.loteriabogota.loteriabackend.domain.model.PaymentOrder;
7   + import co.gov.loteriabogota.loteriabackend.domain.model.User;
8   + import co.gov.loteriabogota.loteriabackend.domain.port.in.LotteryUseCase;
9   + import co.gov.loteriabogota.loteriabackend.domain.port.in.TicketUseCase;
10  + import co.gov.loteriabogota.loteriabackend.domain.port.out.*;
```

```
11  + import co.gov.loteriabogota.loteriabackend.domain.vo.Email;
12  + import co.gov.loteriabogota.loteriabackend.domain.vo.Money;
13  + import co.gov.loteriabogota.loteriabackend.domain.vo.PhoneNumber;
14  + import org.junit.jupiter.api.BeforeEach;
15  + import org.junit.jupiter.api.Test;
16  + import org.junit.jupiter.api.extension.ExtendWith;
17  + import org.mockito.InjectMocks;
18  + import org.mockito.Mock;
19  + import org.mockito.junit.jupiter.MockitoExtension;
20  +
21  + import java.util.List;
22  + import java.util.Optional;
23  + import java.util.UUID;
24  +
25  + import static org.assertj.core.api.Assertions.*;
26  + import static org.mockito.ArgumentMatchers.*;
27  + import static org.mockito.Mockito.*;
28  +
29  + @ExtendWith(MockitoExtension.class)
30  + class PaymentServiceTest {
31  +
32  +     @Mock
33  +     private PaymentOrderRepository paymentOrderRepository;
34  +
35  +     @Mock
36  +     private TicketRepository ticketRepository;
37  +
38  +     @Mock
39  +     private UserRepository userRepository;
40  +
41  +     @Mock
42  +     private LotteryUseCase lotteryUseCase;
43  +
44  +     @Mock
45  +     private PaymentGateway paymentGateway;
46  +
47  +     @InjectMocks
48  +     private PaymentService paymentService;
49  +
50  +     private Email email;
51  +     private Lottery lottery;
52  +     private UUID lotteryId;
```

```
53  +     private User user;
54  +
55  +     @BeforeEach
56  +     void setUp() {
57  +         email = new Email("test@example.com");
58  +         lotteryId = UUID.randomUUID();
59  +
60  +         lottery = new Lottery();
61  +         lottery.setId(lotteryId);
62  +         lottery.setName("Loteria de Bogota");
63  +         lottery.setPricePerFraction(Money.cop(5000));
64  +         lottery.setTotalFractions(10);
65  +
66  +         user = User.createBasic(email);
67  +     }
68  +
69  +     @Test
70  +     void createPayment_withEmail_createsOrderAndTickets() {
71  +         when(userRepository.findByEmail(email)).thenReturn(Optional.of(user));
72  +         when(lotteryUseCase.getLotteryById(lotteryId)).thenReturn(lottery);
73  +         when(paymentGateway.createSession(anyString(), anyString(), any(),
    anyString(),
74  +                 anyString(), isNull(), anyString(), anyString(), anyString()))
75  +                 .thenReturn(new PaymentGateway.SessionResult(12345L,
    "https://pay.example.com"));
76  +         when(paymentOrderRepository.save(any(PaymentOrder.class))).thenAnswer(inv
    -> inv.getArgument(0));
77  +         when(ticketRepository.save(any(LotteryTicket.class))).thenAnswer(inv ->
    inv.getArgument(0));
78  +
79  +         List<TicketUseCase.TicketItem> items = List.of(
80  +                 new TicketUseCase.TicketItem(lotteryId, "1234", "001", 2)
81  +         );
82  +         PaymentOrder result = paymentService.createPayment(email, null, items,
83  +                 "https://return.com", "127.0.0.1", "Mozilla");
84  +
85  +
    assertThat(result.getStatus()).isEqualTo(PaymentOrder.PaymentStatus.PENDING);
86  +         assertThat(result.getPlacetopayRequestId()).isEqualTo(12345L);
87  +
    assertThat(result.getPlacetopayProcessUrl()).isEqualTo("https://pay.example.com");
88  +         verify(ticketRepository).save(any(LotteryTicket.class));
```

```
89  +     }
90  +
91  +     @Test
92  +     void createPayment_withPhone_createsUserFromPhone() {
93  +         PhoneNumber phone = new PhoneNumber("+573001234567");
94  +         User phoneUser = User.createBasicFromPhone(phone);
95  +         when(userRepository.findByPhone(phone)).thenReturn(Optional.empty());
96  +         when(userRepository.save(any(User.class))).thenReturn(phoneUser);
97  +         when(lotteryUseCase.getLotteryById(lotteryId)).thenReturn(lottery);
98  +         when(paymentGateway.createSession(anyString(), anyString(), any(),
    anyString(),
99  +                 anyString(), anyString(), anyString(), anyString(), anyString()))
100 +                 .thenReturn(new PaymentGateway.SessionResult(12345L,
    "https://pay.example.com"));
101 +         when(paymentOrderRepository.save(any(PaymentOrder.class))).thenAnswer(inv
    -> inv.getArgument(0));
102 +         when(ticketRepository.save(any(LotteryTicket.class))).thenAnswer(inv ->
    inv.getArgument(0));
103 +
104 +         List<TicketUseCase.TicketItem> items = List.of(
105 +                 new TicketUseCase.TicketItem(lotteryId, "1234", "001", 2)
106 +         );
107 +         PaymentOrder result = paymentService.createPayment(null, phone, items,
108 +                 "https://return.com", "127.0.0.1", "Mozilla");
109 +
110 +         assertThat(result.getPhone()).isEqualTo("+573001234567");
111 +         verify(userRepository).save(any(User.class));
112 +     }
113 +
114 +     @Test
115 +     void createPayment_emptyItems_throwsDomainException() {
116 +         assertThatThrownBy(() -> paymentService.createPayment(email, null,
    List.of(),
117 +                 "https://return.com", "127.0.0.1", "Mozilla"))
118 +                 .isInstanceOf(DomainException.class)
119 +                 .hasMessageContaining("al menos un billete");
120 +     }
121 +
122 +     @Test
123 +     void createPayment_noEmailNoPhone_throwsDomainException() {
124 +         List<TicketUseCase.TicketItem> items = List.of(
125 +                 new TicketUseCase.TicketItem(lotteryId, "1234", "001", 2)
```

```
126 +         );
127 +
128 +         assertThatThrownBy(() -> paymentService.createPayment(null, null, items,
129 +                 "https://return.com", "127.0.0.1", "Mozilla"))
130 +                 .isInstanceOf(DomainException.class)
131 +                 .hasMessageContaining("correo electronico o un numero de
    celular");
132 +     }
133 +
134 +     @Test
135 +     void createPayment_invalidNumber_throwsDomainException() {
136 +         when(userRepository.findByEmail(email)).thenReturn(Optional.of(user));
137 +         when(lotteryUseCase.getLotteryById(lotteryId)).thenReturn(lottery);
138 +
139 +         List<TicketUseCase.TicketItem> items = List.of(
140 +                 new TicketUseCase.TicketItem(lotteryId, "12", "001", 2)
141 +         );
142 +
143 +         assertThatThrownBy(() -> paymentService.createPayment(email, null, items,
144 +                 "https://return.com", "127.0.0.1", "Mozilla"))
145 +                 .isInstanceOf(DomainException.class)
146 +                 .hasMessageContaining("4 digitos");
147 +     }
148 +
149 +     @Test
150 +     void checkPaymentStatus_approved_updatesOrder() {
151 +         PaymentOrder order = new PaymentOrder(user.getId(), email.value(), "LB-
    123", Money.cop(10000));
152 +         order.setPlacetopayRequestId(12345L);
153 +         when(paymentOrderRepository.findByReference("LB-
    123")).thenReturn(Optional.of(order));
154 +         when(paymentGateway.querySession(12345L))
155 +                 .thenReturn(new PaymentGateway.SessionStatus("APPROVED", null,
    null));
156 +         when(paymentOrderRepository.save(any(PaymentOrder.class))).thenAnswer(inv
    -> inv.getArgument(0));
157 +
158 +         PaymentOrder result = paymentService.checkPaymentStatus("LB-123");
159 +
160 +
    assertThat(result.getStatus()).isEqualTo(PaymentOrder.PaymentStatus.APPROVED);
```

```
161 +         verify(ticketRepository).updateStatusByPaymentOrderId(order.getId(),
    LotteryTicket.TicketStatus.CONFIRMED);
162 +     }
163 +
164 +     @Test
165 +     void checkPaymentStatus_rejected_updatesOrder() {
166 +         PaymentOrder order = new PaymentOrder(user.getId(), email.value(), "LB-
    123", Money.cop(10000));
167 +         order.setPlacetopayRequestId(12345L);
168 +         when(paymentOrderRepository.findByReference("LB-
    123")).thenReturn(Optional.of(order));
169 +         when(paymentGateway.querySession(12345L))
170 +                 .thenReturn(new PaymentGateway.SessionStatus("REJECTED", null,
    null));
171 +         when(paymentOrderRepository.save(any(PaymentOrder.class))).thenAnswer(inv
    -> inv.getArgument(0));
172 +
173 +         PaymentOrder result = paymentService.checkPaymentStatus("LB-123");
174 +
175 +
    assertThat(result.getStatus()).isEqualTo(PaymentOrder.PaymentStatus.REJECTED);
176 +         verify(ticketRepository).updateStatusByPaymentOrderId(order.getId(),
    LotteryTicket.TicketStatus.CANCELLED);
177 +     }
178 +
179 +     @Test
180 +     void checkPaymentStatus_alreadyApproved_skipsGatewayQuery() {
181 +         PaymentOrder order = new PaymentOrder(user.getId(), email.value(), "LB-
    123", Money.cop(10000));
182 +         order.setStatus(PaymentOrder.PaymentStatus.APPROVED);
183 +         when(paymentOrderRepository.findByReference("LB-
    123")).thenReturn(Optional.of(order));
184 +
185 +         PaymentOrder result = paymentService.checkPaymentStatus("LB-123");
186 +
187 +
    assertThat(result.getStatus()).isEqualTo(PaymentOrder.PaymentStatus.APPROVED);
188 +         verifyNoInteractions(paymentGateway);
189 +     }
190 +
191 +     @Test
192 +     void checkPaymentStatus_orderNotFound_throwsDomainException() {
```

```
193 +
        when(paymentOrderRepository.findByReference("INVALID")).thenReturn(Optional.empty(
        ));
194 +
195 +         assertThatThrownBy(() -> paymentService.checkPaymentStatus("INVALID"))
196 +                 .isInstanceOf(DomainException.class)
197 +                 .hasMessageContaining("no encontrada");
198 +     }
199 +
200 +     @Test
201 +     void checkPaymentStatus_noRequestId_throwsDomainException() {
202 +         PaymentOrder order = new PaymentOrder(user.getId(), email.value(), "LB-
        123", Money.cop(10000));
203 +         // no setPlacetopayRequestId
204 +         when(paymentOrderRepository.findByReference("LB-
        123")).thenReturn(Optional.of(order));
205 +
206 +         assertThatThrownBy(() -> paymentService.checkPaymentStatus("LB-123"))
207 +                 .isInstanceOf(DomainException.class)
208 +                 .hasMessageContaining("sin sesion de pago");
209 +     }
210 + }
```

…ain/service/PhoneVerificationServiceTest.java

```
••• @@ -0,0 +1,158 @@
1   + package co.gov.loteriabogota.loteriabackend.domain.service;
2   +
3   + import
        co.gov.loteriabogota.loteriabackend.domain.port.in.PhoneVerificationUseCase;
4   + import co.gov.loteriabogota.loteriabackend.domain.port.out.PhoneVerificationStore;
5   + import co.gov.loteriabogota.loteriabackend.domain.port.out.SmsGateway;
6   + import co.gov.loteriabogota.loteriabackend.domain.vo.PhoneNumber;
7   + import org.junit.jupiter.api.BeforeEach;
8   + import org.junit.jupiter.api.Test;
9   + import org.junit.jupiter.api.extension.ExtendWith;
10  + import org.mockito.ArgumentCaptor;
11  + import org.mockito.InjectMocks;
12  + import org.mockito.Mock;
13  + import org.mockito.junit.jupiter.MockitoExtension;
14  + import org.springframework.test.util.ReflectionTestUtils;
15  +
```

```
16  + import java.time.Instant;
17  + import java.util.Optional;
18  +
19  + import static org.assertj.core.api.Assertions.*;
20  + import static org.mockito.ArgumentMatchers.*;
21  + import static org.mockito.Mockito.*;
22  +
23  + @ExtendWith(MockitoExtension.class)
24  + class PhoneVerificationServiceTest {
25  +
26  +     @Mock
27  +     private PhoneVerificationStore store;
28  +
29  +     @Mock
30  +     private SmsGateway smsGateway;
31  +
32  +     @InjectMocks
33  +     private PhoneVerificationService service;
34  +
35  +     private PhoneNumber phone;
36  +
37  +     @BeforeEach
38  +     void setUp() {
39  +         phone = new PhoneNumber("+573001234567");
40  +         ReflectionTestUtils.setField(service, "ttlSeconds", 300L);
41  +         ReflectionTestUtils.setField(service, "maxAttempts", 5);
42  +     }
43  +
44  +     @Test
45  +     void requestSmsOtp_savesCodeAndSendsSms() {
46  +         service.requestSmsOtp(phone);
47  +
48  +         verify(store).savePending(eq(phone), anyString(), any(Instant.class));
49  +         ArgumentCaptor<String> msgCaptor = ArgumentCaptor.forClass(String.class);
50  +         verify(smsGateway).send(eq(phone), msgCaptor.capture());
51  +         assertThat(msgCaptor.getValue()).contains("codigo de verificacion");
52  +     }
53  +
54  +     @Test
55  +     void verifySmsOtp_validCode_returnsVerificationToken() {
56  +         // We need to compute the sha256 of "123456" to match
57  +         String code = "123456";
```

```
58  +          String codeHash = sha256(code);
59  +          PhoneVerificationStore.PendingOtp pending =
60  +                  new PhoneVerificationStore.PendingOtp("token-123", codeHash,
61  +                          Instant.now().plusSeconds(300), false, 0);
62  +          when(store.findLatestPending(phone)).thenReturn(Optional.of(pending));
63  +
64  +          PhoneVerificationUseCase.VerifyResult result = service.verifySmsOtp(phone,
      code);
65  +
66  +          assertThat(result.verificationToken()).isEqualTo("token-123");
67  +          verify(store).markVerified("token-123");
68  +      }
69  +
70  +      @Test
71  +      void verifySmsOtp_wrongCode_throwsAndIncrementsAttempts() {
72  +          PhoneVerificationStore.PendingOtp pending =
73  +                  new PhoneVerificationStore.PendingOtp("token-123", "wrong-hash",
74  +                          Instant.now().plusSeconds(300), false, 0);
75  +          when(store.findLatestPending(phone)).thenReturn(Optional.of(pending));
76  +
77  +          assertThatThrownBy(() -> service.verifySmsOtp(phone, "999999"))
78  +                  .isInstanceOf(IllegalArgumentException.class)
79  +                  .hasMessageContaining("invalido");
80  +
81  +          verify(store).incrementAttempts("token-123");
82  +      }
83  +
84  +      @Test
85  +      void verifySmsOtp_expired_throwsException() {
86  +          PhoneVerificationStore.PendingOtp pending =
87  +                  new PhoneVerificationStore.PendingOtp("token-123", "hash",
88  +                          Instant.now().minusSeconds(10), false, 0);
89  +          when(store.findLatestPending(phone)).thenReturn(Optional.of(pending));
90  +
91  +          assertThatThrownBy(() -> service.verifySmsOtp(phone, "123456"))
92  +                  .isInstanceOf(IllegalArgumentException.class)
93  +                  .hasMessageContaining("expiro");
94  +      }
95  +
96  +      @Test
97  +      void verifySmsOtp_alreadyUsed_throwsException() {
98  +          PhoneVerificationStore.PendingOtp pending =
```

```
99  +                new PhoneVerificationStore.PendingOtp("token-123", "hash",
100 +                        Instant.now().plusSeconds(300), true, 0);
101 +        when(store.findLatestPending(phone)).thenReturn(Optional.of(pending));
102 +
103 +        assertThatThrownBy(() -> service.verifySmsOtp(phone, "123456"))
104 +                .isInstanceOf(IllegalArgumentException.class)
105 +                .hasMessageContaining("ya fue usado");
106 +    }
107 +
108 +    @Test
109 +    void verifySmsOtp_tooManyAttempts_throwsException() {
110 +        PhoneVerificationStore.PendingOtp pending =
111 +                new PhoneVerificationStore.PendingOtp("token-123", "hash",
112 +                        Instant.now().plusSeconds(300), false, 5);
113 +        when(store.findLatestPending(phone)).thenReturn(Optional.of(pending));
114 +
115 +        assertThatThrownBy(() -> service.verifySmsOtp(phone, "123456"))
116 +                .isInstanceOf(IllegalArgumentException.class)
117 +                .hasMessageContaining("Demasiados intentos");
118 +    }
119 +
120 +    @Test
121 +    void verifySmsOtp_noPending_throwsException() {
122 +        when(store.findLatestPending(phone)).thenReturn(Optional.empty());
123 +
124 +        assertThatThrownBy(() -> service.verifySmsOtp(phone, "123456"))
125 +                .isInstanceOf(IllegalArgumentException.class)
126 +                .hasMessageContaining("No hay un codigo pendiente");
127 +    }
128 +
129 +    @Test
130 +    void isVerified_delegatesToStore() {
131 +        when(store.isVerifiedTokenForPhone("token-123", phone)).thenReturn(true);
132 +
133 +        assertThat(service.isVerified(phone, "token-123")).isTrue();
134 +    }
135 +
136 +    @Test
137 +    void isVerified_nullToken_returnsFalse() {
138 +        assertThat(service.isVerified(phone, null)).isFalse();
139 +    }
140 +
```

```
141 +     @Test
142 +     void isVerified_blankToken_returnsFalse() {
143 +         assertThat(service.isVerified(phone, "  ")).isFalse();
144 +     }
145 +
146 +     // Helper to compute SHA-256 (same as the service)
147 +     private String sha256(String s) {
148 +         try {
149 +             java.security.MessageDigest md =
    java.security.MessageDigest.getInstance("SHA-256");
150 +             byte[] digest =
    md.digest(s.getBytes(java.nio.charset.StandardCharsets.UTF_8));
151 +             StringBuilder sb = new StringBuilder();
152 +             for (byte b : digest) sb.append(String.format("%02x", b));
153 +             return sb.toString();
154 +         } catch (Exception ex) {
155 +             throw new RuntimeException(ex);
156 +         }
157 +     }
158 + }
```

…backend/domain/service/TicketServiceTest.java

```
    @@ -0,0 +1,174 @@
1   + package co.gov.loteriabogota.loteriabackend.domain.service;
2   +
3   + import co.gov.loteriabogota.loteriabackend.domain.exception.DomainException;
4   + import co.gov.loteriabogota.loteriabackend.domain.model.Lottery;
5   + import co.gov.loteriabogota.loteriabackend.domain.model.LotteryTicket;
6   + import co.gov.loteriabogota.loteriabackend.domain.model.User;
7   + import co.gov.loteriabogota.loteriabackend.domain.port.in.LotteryUseCase;
8   + import co.gov.loteriabogota.loteriabackend.domain.port.in.TicketUseCase;
9   + import co.gov.loteriabogota.loteriabackend.domain.port.out.TicketRepository;
10  + import co.gov.loteriabogota.loteriabackend.domain.port.out.UserRepository;
11  + import co.gov.loteriabogota.loteriabackend.domain.vo.Email;
12  + import co.gov.loteriabogota.loteriabackend.domain.vo.Money;
13  + import org.junit.jupiter.api.BeforeEach;
14  + import org.junit.jupiter.api.Test;
15  + import org.junit.jupiter.api.extension.ExtendWith;
16  + import org.mockito.InjectMocks;
17  + import org.mockito.Mock;
18  + import org.mockito.junit.jupiter.MockitoExtension;
```

```
19  +
20  + import java.util.List;
21  + import java.util.Optional;
22  + import java.util.UUID;
23  +
24  + import static org.assertj.core.api.Assertions.*;
25  + import static org.mockito.ArgumentMatchers.any;
26  + import static org.mockito.Mockito.*;
27  +
28  + @ExtendWith(MockitoExtension.class)
29  + class TicketServiceTest {
30  +
31  +     @Mock
32  +     private TicketRepository ticketRepository;
33  +
34  +     @Mock
35  +     private LotteryUseCase lotteryUseCase;
36  +
37  +     @Mock
38  +     private UserRepository userRepository;
39  +
40  +     @InjectMocks
41  +     private TicketService ticketService;
42  +
43  +     private Lottery lottery;
44  +     private UUID lotteryId;
45  +     private UUID userId;
46  +
47  +     @BeforeEach
48  +     void setUp() {
49  +         lotteryId = UUID.randomUUID();
50  +         userId = UUID.randomUUID();
51  +
52  +         lottery = new Lottery();
53  +         lottery.setId(lotteryId);
54  +         lottery.setName("Loteria de Bogota");
55  +         lottery.setPricePerFraction(Money.cop(5000));
56  +         lottery.setTotalFractions(10);
57  +         lottery.setActive(true);
58  +     }
59  +
60  +     @Test
```

```
61  +     void purchaseTicket_validData_createsTicket() {
62  +         when(lotteryUseCase.getLotteryById(lotteryId)).thenReturn(lottery);
63  +         when(ticketRepository.save(any(LotteryTicket.class))).thenAnswer(inv ->
        inv.getArgument(0));
64  +
65  +         LotteryTicket result = ticketService.purchaseTicket(userId, lotteryId,
        "1234", "001", 3);
66  +
67  +         assertThat(result.getUserId()).isEqualTo(userId);
68  +         assertThat(result.getLotteryId()).isEqualTo(lotteryId);
69  +         assertThat(result.getNumber()).isEqualTo("1234");
70  +         assertThat(result.getSeries()).isEqualTo("001");
71  +         assertThat(result.getFractions()).isEqualTo(3);
72  +         assertThat(result.getTotalPrice().amount().longValue()).isEqualTo(15000);
73  +
        assertThat(result.getStatus()).isEqualTo(LotteryTicket.TicketStatus.PENDING);
74  +     }
75  +
76  +     @Test
77  +     void purchaseTicket_invalidNumber_throwsDomainException() {
78  +         when(lotteryUseCase.getLotteryById(lotteryId)).thenReturn(lottery);
79  +
80  +         assertThatThrownBy(() -> ticketService.purchaseTicket(userId, lotteryId,
        "12", "001", 3))
81  +                 .isInstanceOf(DomainException.class)
82  +                 .hasMessageContaining("4 digitos");
83  +     }
84  +
85  +     @Test
86  +     void purchaseTicket_invalidSeries_throwsDomainException() {
87  +         when(lotteryUseCase.getLotteryById(lotteryId)).thenReturn(lottery);
88  +
89  +         assertThatThrownBy(() -> ticketService.purchaseTicket(userId, lotteryId,
        "1234", "01", 3))
90  +                 .isInstanceOf(DomainException.class)
91  +                 .hasMessageContaining("3 digitos");
92  +     }
93  +
94  +     @Test
95  +     void purchaseTicket_tooManyFractions_throwsDomainException() {
96  +         when(lotteryUseCase.getLotteryById(lotteryId)).thenReturn(lottery);
97  +
```

```
98  +         assertThatThrownBy(() -> ticketService.purchaseTicket(userId, lotteryId,
    "1234", "001", 15))
99  +                 .isInstanceOf(DomainException.class)
100 +                 .hasMessageContaining("fracciones");
101 +     }
102 +
103 +     @Test
104 +     void purchaseTicket_zeroFractions_throwsDomainException() {
105 +         when(lotteryUseCase.getLotteryById(lotteryId)).thenReturn(lottery);
106 +
107 +         assertThatThrownBy(() -> ticketService.purchaseTicket(userId, lotteryId,
    "1234", "001", 0))
108 +                 .isInstanceOf(DomainException.class)
109 +                 .hasMessageContaining("fracciones");
110 +     }
111 +
112 +     @Test
113 +     void checkout_existingUser_usesExistingUser() {
114 +         Email email = new Email("test@example.com");
115 +         User existingUser = User.createBasic(email);
116 +
    when(userRepository.findByEmail(email)).thenReturn(Optional.of(existingUser));
117 +         when(lotteryUseCase.getLotteryById(lotteryId)).thenReturn(lottery);
118 +         when(ticketRepository.save(any(LotteryTicket.class))).thenAnswer(inv ->
    inv.getArgument(0));
119 +
120 +         List<TicketUseCase.TicketItem> items = List.of(
121 +                 new TicketUseCase.TicketItem(lotteryId, "1234", "001", 2)
122 +         );
123 +         List<LotteryTicket> result = ticketService.checkout(email, items);
124 +
125 +         assertThat(result).hasSize(1);
126 +         verify(userRepository, never()).save(any());
127 +     }
128 +
129 +     @Test
130 +     void checkout_newUser_createsUser() {
131 +         Email email = new Email("nuevo@example.com");
132 +         User newUser = User.createBasic(email);
133 +         when(userRepository.findByEmail(email)).thenReturn(Optional.empty());
134 +         when(userRepository.save(any(User.class))).thenReturn(newUser);
135 +         when(lotteryUseCase.getLotteryById(lotteryId)).thenReturn(lottery);
```

```
136 +         when(ticketRepository.save(any(LotteryTicket.class))).thenAnswer(inv ->
    inv.getArgument(0));
137 +
138 +         List<TicketUseCase.TicketItem> items = List.of(
139 +                 new TicketUseCase.TicketItem(lotteryId, "1234", "001", 2)
140 +         );
141 +         List<LotteryTicket> result = ticketService.checkout(email, items);
142 +
143 +         assertThat(result).hasSize(1);
144 +         verify(userRepository).save(any(User.class));
145 +     }
146 +
147 +     @Test
148 +     void checkout_emptyItems_throwsDomainException() {
149 +         Email email = new Email("test@example.com");
150 +
151 +         assertThatThrownBy(() -> ticketService.checkout(email, List.of()))
152 +                 .isInstanceOf(DomainException.class)
153 +                 .hasMessageContaining("al menos un billete");
154 +     }
155 +
156 +     @Test
157 +     void checkout_nullItems_throwsDomainException() {
158 +         Email email = new Email("test@example.com");
159 +
160 +         assertThatThrownBy(() -> ticketService.checkout(email, null))
161 +                 .isInstanceOf(DomainException.class)
162 +                 .hasMessageContaining("al menos un billete");
163 +     }
164 +
165 +     @Test
166 +     void getUserTickets_delegatesToRepository() {
167 +         when(ticketRepository.findByUserId(userId)).thenReturn(List.of());
168 +
169 +         List<LotteryTicket> result = ticketService.getUserTickets(userId);
170 +
171 +         assertThat(result).isEmpty();
172 +         verify(ticketRepository).findByUserId(userId);
173 +     }
174 + }
```

…iabackend/domain/service/UserServiceTest.java

```
@@ -0,0 +1,93 @@
+ package co.gov.loteriabogota.loteriabackend.domain.service;
+
+ import co.gov.loteriabogota.loteriabackend.domain.exception.UserNotFoundException;
+ import co.gov.loteriabogota.loteriabackend.domain.model.User;
+ import co.gov.loteriabogota.loteriabackend.domain.port.out.UserRepository;
+ import co.gov.loteriabogota.loteriabackend.domain.vo.Email;
+ 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.Mock;
+ import org.mockito.junit.jupiter.MockitoExtension;
+
+ import java.time.LocalDate;
+ import java.util.Optional;
+ import java.util.UUID;
+
+ import static org.assertj.core.api.Assertions.*;
+ import static org.mockito.ArgumentMatchers.any;
+ import static org.mockito.Mockito.*;
+
+ @ExtendWith(MockitoExtension.class)
+ class UserServiceTest {
+
+     @Mock
+     private UserRepository userRepository;
+
+     @InjectMocks
+     private UserService userService;
+
+     private User user;
+     private UUID userId;
+
+     @BeforeEach
+     void setUp() {
+         userId = UUID.randomUUID();
+         user = new User(userId, new Email("test@example.com"),
co.gov.loteriabogota.loteriabackend.domain.model.KycLevel.BASIC);
```

```
38  +     }
39  +
40  +     @Test
41  +     void getUserById_found() {
42  +         when(userRepository.findById(userId)).thenReturn(Optional.of(user));
43  +
44  +         User result = userService.getUserById(userId);
45  +
46  +         assertThat(result.getId()).isEqualTo(userId);
47  +     }
48  +
49  +     @Test
50  +     void getUserById_notFound_throwsException() {
51  +         UUID unknownId = UUID.randomUUID();
52  +         when(userRepository.findById(unknownId)).thenReturn(Optional.empty());
53  +
54  +         assertThatThrownBy(() -> userService.getUserById(unknownId))
55  +                 .isInstanceOf(UserNotFoundException.class);
56  +     }
57  +
58  +     @Test
59  +     void updateProfile_setsDocumentAndBirthDateAndPhone() {
60  +         when(userRepository.findById(userId)).thenReturn(Optional.of(user));
61  +         when(userRepository.save(any(User.class))).thenAnswer(inv ->
        inv.getArgument(0));
62  +
63  +         User result = userService.updateProfile(userId, "12345678",
        LocalDate.of(1990, 5, 15), "+573001234567");
64  +
65  +         assertThat(result.getDocumentId().type()).isEqualTo("CC");
66  +         assertThat(result.getDocumentId().number()).isEqualTo("12345678");
67  +         assertThat(result.getBirthDate()).isEqualTo(LocalDate.of(1990, 5, 15));
68  +         assertThat(result.getPhone()).isEqualTo("+573001234567");
69  +     }
70  +
71  +     @Test
72  +     void updateProfile_partialUpdate_onlyDocument() {
73  +         when(userRepository.findById(userId)).thenReturn(Optional.of(user));
74  +         when(userRepository.save(any(User.class))).thenAnswer(inv ->
        inv.getArgument(0));
75  +
76  +         User result = userService.updateProfile(userId, "12345678", null, null);
```

```
77  +
78  +         assertThat(result.getDocumentId().number()).isEqualTo("12345678");
79  +         assertThat(result.getBirthDate()).isNull();
80  +         assertThat(result.getPhone()).isNull();
81  +     }
82  +
83  +     @Test
84  +     void updateProfile_noChanges() {
85  +         when(userRepository.findById(userId)).thenReturn(Optional.of(user));
86  +         when(userRepository.save(any(User.class))).thenAnswer(inv ->
      inv.getArgument(0));
87  +
88  +         User result = userService.updateProfile(userId, null, null, null);
89  +
90  +         assertThat(result.getDocumentId()).isNull();
91  +         assertThat(result.getUpdatedAt()).isNotNull();
92  +     }
93  + }
```

…oteriabackend/domain/vo/ValueObjectsTest.java

```
    @@ -0,0 +1,140 @@
1   + package co.gov.loteriabogota.loteriabackend.domain.vo;
2   +
3   + import org.junit.jupiter.api.Test;
4   +
5   + import java.math.BigDecimal;
6   +
7   + import static org.assertj.core.api.Assertions.*;
8   +
9   + class ValueObjectsTest {
10  +
11  +     // --- Email ---
12  +
13  +     @Test
14  +     void email_validEmail_creates() {
15  +         Email email = new Email("test@example.com");
16  +         assertThat(email.value()).isEqualTo("test@example.com");
17  +     }
18  +
19  +     @Test
20  +     void email_invalidFormat_throws() {
```

```
21  +          assertThatThrownBy(() -> new Email("not-an-email"))
22  +                  .isInstanceOf(IllegalArgumentException.class)
23  +                  .hasMessageContaining("Invalid email");
24  +      }
25  +
26  +      @Test
27  +      void email_null_throws() {
28  +          assertThatThrownBy(() -> new Email(null))
29  +                  .isInstanceOf(IllegalArgumentException.class);
30  +      }
31  +
32  +      // --- PhoneNumber ---
33  +
34  +      @Test
35  +      void phoneNumber_validWithPlus_creates() {
36  +          PhoneNumber phone = new PhoneNumber("+573001234567");
37  +          assertThat(phone.value()).isEqualTo("+573001234567");
38  +      }
39  +
40  +      @Test
41  +      void phoneNumber_validWithoutPlus_creates() {
42  +          PhoneNumber phone = new PhoneNumber("3001234567");
43  +          assertThat(phone.value()).isEqualTo("3001234567");
44  +      }
45  +
46  +      @Test
47  +      void phoneNumber_tooShort_throws() {
48  +          assertThatThrownBy(() -> new PhoneNumber("123"))
49  +                  .isInstanceOf(IllegalArgumentException.class)
50  +                  .hasMessageContaining("Invalid phone");
51  +      }
52  +
53  +      @Test
54  +      void phoneNumber_withLetters_throws() {
55  +          assertThatThrownBy(() -> new PhoneNumber("+57abc1234567"))
56  +                  .isInstanceOf(IllegalArgumentException.class);
57  +      }
58  +
59  +      @Test
60  +      void phoneNumber_null_throws() {
61  +          assertThatThrownBy(() -> new PhoneNumber(null))
62  +                  .isInstanceOf(IllegalArgumentException.class);
```

```
63  +     }
64  +
65  +     // --- Money ---
66  +
67  +     @Test
68  +     void money_copFromBigDecimal_creates() {
69  +         Money money = Money.cop(BigDecimal.valueOf(5000));
70  +         assertThat(money.amount()).isEqualByComparingTo(BigDecimal.valueOf(5000));
71  +         assertThat(money.currency()).isEqualTo(Money.COP);
72  +     }
73  +
74  +     @Test
75  +     void money_copFromLong_creates() {
76  +         Money money = Money.cop(5000);
77  +         assertThat(money.amount()).isEqualByComparingTo(BigDecimal.valueOf(5000));
78  +     }
79  +
80  +     @Test
81  +     void money_negativeAmount_throws() {
82  +         assertThatThrownBy(() -> Money.cop(BigDecimal.valueOf(-1)))
83  +                 .isInstanceOf(IllegalArgumentException.class)
84  +                 .hasMessageContaining("non-negative");
85  +     }
86  +
87  +     @Test
88  +     void money_nullAmount_throws() {
89  +         assertThatThrownBy(() -> new Money(null, Money.COP))
90  +                 .isInstanceOf(IllegalArgumentException.class);
91  +     }
92  +
93  +     @Test
94  +     void money_nullCurrency_throws() {
95  +         assertThatThrownBy(() -> new Money(BigDecimal.TEN, null))
96  +                 .isInstanceOf(IllegalArgumentException.class)
97  +                 .hasMessageContaining("Currency");
98  +     }
99  +
100 +     @Test
101 +     void money_zeroAmount_valid() {
102 +         Money money = Money.cop(BigDecimal.ZERO);
103 +         assertThat(money.amount()).isEqualByComparingTo(BigDecimal.ZERO);
104 +     }
```

```
105 +
106 +     // --- DocumentId ---
107 +
108 +     @Test
109 +     void documentId_valid_creates() {
110 +         DocumentId doc = new DocumentId("CC", "12345678");
111 +         assertThat(doc.type()).isEqualTo("CC");
112 +         assertThat(doc.number()).isEqualTo("12345678");
113 +     }
114 +
115 +     @Test
116 +     void documentId_blankType_throws() {
117 +         assertThatThrownBy(() -> new DocumentId("", "12345678"))
118 +                 .isInstanceOf(IllegalArgumentException.class)
119 +                 .hasMessageContaining("type");
120 +     }
121 +
122 +     @Test
123 +     void documentId_blankNumber_throws() {
124 +         assertThatThrownBy(() -> new DocumentId("CC", ""))
125 +                 .isInstanceOf(IllegalArgumentException.class)
126 +                 .hasMessageContaining("number");
127 +     }
128 +
129 +     @Test
130 +     void documentId_nullType_throws() {
131 +         assertThatThrownBy(() -> new DocumentId(null, "12345678"))
132 +                 .isInstanceOf(IllegalArgumentException.class);
133 +     }
134 +
135 +     @Test
136 +     void documentId_nullNumber_throws() {
137 +         assertThatThrownBy(() -> new DocumentId("CC", null))
138 +                 .isInstanceOf(IllegalArgumentException.class);
139 +     }
140 + }
```

Comments 0

Lock conversation

Comment

Subscribe You're not receiving notifications from this thread.