Критерий 1.3 — код модулей автоматизированного тестирования

Автоматизированное тестирование

Тестирование программного обеспечения

В ходе работы было организовано тестирование Wallenc на нескольких уровнях: модульные автоматические тесты (JUnit, каталог `src/test` каждого Gradle-модуля), инструментальные тесты (`src/androidTest`), а также ручные функциональные и UI-прогоны. Программа и методика испытаний приведены в приложении Б пояснительной записки.

1.2 Автоматизированное тестирование (код модулей)

В проекте реализовано 68 автоматических unit-тестов в пяти модулях (`:domain` — 12, `:domain-vault` — 10, `:usecases` — 25, `:ui` — 15, `:task-runtime` — 6). Тесты выполняются на JVM при сборке (`./gradlew test`). Инструментальные тесты размещены в `src/androidTest` соответствующих модулей.

1.2.1 Листинги исходного кода автотестов

Ниже — листинги файлов `src/test` и `src/androidTest` (при сборке PDF читаются из дерева проекта Wallenc, по тому же принципу, что приложение А пояснительной записки).

1.3 Модуль :domain

Автотест domain/src/test/java/com/github/nullptroma/wallenc/domain/EncryptorTest.kt

```kotlin
package com.github.nullptroma.wallenc.domain

import com.github.nullptroma.wallenc.domain.datatypes.EncryptKey
import com.github.nullptroma.wallenc.domain.encrypt.Encryptor
import junit.framework.TestCase.assertEquals
import junit.framework.TestCase.fail
import org.hamcrest.MatcherAssert.assertThat
import org.hamcrest.core.Is
import org.hamcrest.core.IsEqual
import org.hamcrest.core.IsNot
import org.junit.Assert.assertArrayEquals
import org.junit.Test
import java.io.ByteArrayInputStream
import java.io.ByteArrayOutputStream
import java.util.Random

class EncryptorTest {
    val key1 = EncryptKey("key1")
    val key2 = EncryptKey("key2")
    val rnd = Random()

    @Test
    fun `test correct key for StorageEncryptionInfo`() {
        val encInfo = Encryptor.generateEncryptionInfo(key1)
        val res = Encryptor.checkKey(key = key1, encInfo = encInfo)
        assertEquals(true, res)
    }

    @Test
    fun `test incorrect key for StorageEncryptionInfo`() {
        val encInfo = Encryptor.generateEncryptionInfo(key1)
        val res = Encryptor.checkKey(key = key2, encInfo = encInfo)
        assertEquals(false, res)
    }

    @Test
    fun `test string encryption with the same key`() {
        val text = "Hello world, my name is Wallenc!"
        val encryptor = Encryptor(key1.toAesKey())
        val encryptedText = encryptor.encryptString(text)

        val newEncryptor = Encryptor(key1.toAesKey())
        val decryptedText = newEncryptor.decryptString(encryptedText)
```

```
44
45              assertEquals(text, decryptedText)
46          }
47
48          @Test
49          fun `test string encryption with the wrong key`() {
50              val text = "Hello world, my name is Wallenc!"
51              val encryptor = Encryptor(key1.toAesKey())
52              val encryptedText = encryptor.encryptString(text)
53
54              val newEncryptor = Encryptor(key2.toAesKey())
55              try {
56                  newEncryptor.decryptString(encryptedText)
57                  fail("There is not exception on decrypt with wrong key")
58              }
59              catch (e: Exception) {
60                  // good
61              }
62          }
63
64          @Test
65          fun `test bytes encryption with the same key`() {
66              val bytes = ByteArray(512)
67              rnd.nextBytes(bytes)
68              val encryptor = Encryptor(key1.toAesKey())
69              val encryptedBytes = encryptor.encryptBytes(bytes)
70
71              val newEncryptor = Encryptor(key1.toAesKey())
72              val decryptedBytes = newEncryptor.decryptBytes(encryptedBytes)
73
74              assertThat(bytes, IsNot.not(IsEqual.equalTo(encryptedBytes)))
75              assertThat(bytes, Is.`is`(IsEqual.equalTo(decryptedBytes)))
76          }
77
78          @Test
79          fun `test bytes encryption with the wrong key`() {
80              val bytes = ByteArray(512)
81              rnd.nextBytes(bytes)
82              val encryptor = Encryptor(key1.toAesKey())
83              val encryptedBytes = encryptor.encryptBytes(bytes)
84
85              val newEncryptor = Encryptor(key2.toAesKey())
86              try {
87                  newEncryptor.decryptBytes(encryptedBytes)
88                  fail("There is not exception on decrypt with wrong key")
```

```
89              }
90              catch (e: Exception) {
91                  // good
92              }
93          }
94
95          @Test
96          fun `test stream encryption with the same key`() {
97              val dataLen = 1500
98              val origData = ByteArray(dataLen)
99              rnd.nextBytes(origData)
100             val encryptor = Encryptor(key1.toAesKey())
101
102             val streamForEncrypt = ByteArrayOutputStream(dataLen*3)
103             val encryptedStream = encryptor.encryptStream(streamForEncrypt)
104             encryptedStream.write(origData)
105             encryptedStream.close()
106             val encryptedData = streamForEncrypt.toByteArray()
107
108             val newEncryptor = Encryptor(key1.toAesKey())
109             val streamForDecrypt = ByteArrayInputStream(encryptedData)
110             val decryptedStream = newEncryptor.decryptStream(streamForDecrypt)
111             val decryptedData = decryptedStream.readAllBytes()
112
113             assertArrayEquals(origData, decryptedData)
114         }
115
116         @Test
117         fun `test stream encryption with the wrong key`() {
118             val dataLen = 1500
119             val origData = ByteArray(dataLen)
120             rnd.nextBytes(origData)
121             val encryptor = Encryptor(key1.toAesKey())
122
123             val streamForEncrypt = ByteArrayOutputStream(dataLen*3)
124             val encryptedStream = encryptor.encryptStream(streamForEncrypt)
125             encryptedStream.write(origData)
126             encryptedStream.close()
127             val encryptedData = streamForEncrypt.toByteArray()
128
129             val newEncryptor = Encryptor(key2.toAesKey())
130             try {
131                 val streamForDecrypt = ByteArrayInputStream(encryptedData)
132                 val decryptedStream = newEncryptor.decryptStream(streamForDecrypt)
133                 decryptedStream.readAllBytes()
134                 fail("There is not exception on decrypt with wrong key")
```

```
            }
            catch (e: Exception) {
                // good
            }
        }
    }
```

Автотест domain/src/test/java/com/github/nullptroma/wallenc/domain/errors/WallencExceptionMappingTest.kt

```
package com.github.nullptroma.wallenc.domain.errors

import org.junit.Assert.assertSame
import org.junit.Assert.assertTrue
import org.junit.Test
import java.io.FileNotFoundException
import java.io.IOException

class WallencExceptionMappingTest {

    @Test
    fun preservesWallencException() {
        val original = WallencException.Feature.StorageNotFound()
        assertSame(original, original.toWallencException())
    }

    @Test
    fun mapsFileNotFoundException() {
        val mapped = FileNotFoundException("missing").toWallencException()
        assertTrue(mapped is WallencException.Storage.FileNotFound)
    }

    @Test
    fun mapsIOExceptionToIoFailed() {
        val cause = IOException("disk")
        val mapped = cause.toWallencException()
        assertTrue(mapped is WallencException.Network.IoFailed)
        assertSame(cause, (mapped as WallencException.Network.IoFailed).cause)
    }

    @Test
    fun mapsGenericExceptionToUnknown() {
        val cause = Exception("boom")
        val mapped = cause.toWallencException()
        assertTrue(mapped is WallencException.Unknown)
        assertSame(cause, (mapped as WallencException.Unknown).cause)
```

```
37            }
38        }
39
```

1.4 Модуль :domain-vault

Автотест domain-vault/src/test/java/com/github/nullptroma/wallenc/domain/vault/errors/VaultThrowableMappingTest.kt

```kotlin
package com.github.nullptroma.wallenc.domain.vault.errors

import com.github.nullptroma.wallenc.domain.errors.WallencException
import com.github.nullptroma.wallenc.domain.vault.network.yandexdisk.YandexDiskAuthException
import org.junit.Assert.assertEquals
import org.junit.Assert.assertTrue
import org.junit.Test
import okhttp3.ResponseBody.Companion.toResponseBody
import retrofit2.HttpException
import retrofit2.Response
import java.io.FileNotFoundException
import java.io.IOException
import java.net.SocketTimeoutException

class VaultThrowableMappingTest {

    @Test
    fun mapsYandexDiskAuthToAuthFailed() {
        val mapped = YandexDiskAuthException("unauthorized").toVaultWallencException()
        assertTrue(mapped is WallencException.Auth.Failed)
    }

    @Test
    fun mapsHttpExceptionToNetworkHttpFailed() {
        val response = Response.error<String>(401, "".toResponseBody(null))
        val mapped = HttpException(response).toVaultWallencException()
        assertTrue(mapped is WallencException.Network.HttpFailed)
        assertEquals(401, (mapped as WallencException.Network.HttpFailed).statusCode)
    }

    @Test
    fun mapsMissingOAuthTokenIoToTokenMissing() {
        val mapped = IOException("Yandex OAuth token is missing").toVaultWallencException()
        assertTrue(mapped is WallencException.Auth.TokenMissing)
    }

    @Test
    fun mapsSocketTimeoutToOperationTimedOut() {
        val mapped = SocketTimeoutException("timeout").toVaultWallencException()
        assertTrue(mapped is WallencException.Network.OperationTimedOut)
    }

    @Test
```

```
44          fun mapsFileNotFoundToStorageFileNotFound() {
45              val mapped = FileNotFoundException("x").toVaultWallencException()
46              assertTrue(mapped is WallencException.Storage.FileNotFound)
47          }
48
49          @Test
50          fun mapsIllegalStateNotAFile() {
51              val mapped = IllegalStateException("Not a file").toVaultWallencException()
52              assertTrue(mapped is WallencException.Storage.NotAFile)
53          }
54      }
55
```

Автотест domain-vault/src/test/java/com/github/nullptroma/wallenc/domain/vault/network/yandexdisk/repository/YandexDiskRepositoryTest.kt

```
1   package com.github.nullptroma.wallenc.domain.vault.network.yandexdisk.repository
2
3   import com.github.nullptroma.wallenc.domain.vault.network.yandexdisk.YandexDiskAuthException
4   import
    com.github.nullptroma.wallenc.domain.vault.network.yandexdisk.YandexDiskRepositoryTestFactory
5   import kotlinx.coroutines.Dispatchers
6   import kotlinx.coroutines.runBlocking
7   import okhttp3.OkHttpClient
8   import okhttp3.mockwebserver.MockResponse
9   import okhttp3.mockwebserver.MockWebServer
10  import org.junit.After
11  import org.junit.Assert.assertEquals
12  import org.junit.Assert.assertTrue
13  import org.junit.Before
14  import org.junit.Test
15
16  class YandexDiskRepositoryTest {
17
18      private lateinit var server: MockWebServer
19      private lateinit var repository: YandexDiskRepository
20
21      @Before
22      fun setUp() {
23          server = MockWebServer()
24          server.start()
25          val api = YandexDiskRepositoryTestFactory.createApi(server.url("/").toString())
    { "test-token" }
26          repository = YandexDiskRepository(api, OkHttpClient(), Dispatchers.IO)
27      }
28
```

```
29          @After
30          fun tearDown() {
31              server.shutdown()
32          }
33
34          @Test
35          fun diskInfoParsesResponse() = runBlocking {
36              repository.resetCloudApiCallCount()
37              server.enqueue(
38                  MockResponse()
39                      .setResponseCode(200)
40                      .setBody("""{"total_space":1000,"used_space":200,"trash_size":0}""")
41                      .addHeader("Content-Type", "application/json"),
42              )
43              val info = repository.diskInfo()
44              assertEquals(1000L, info.totalSpace)
45              assertEquals(200L, info.usedSpace)
46              assertEquals(1L, repository.cloudApiCallCount())
47          }
48
49          @Test
50          fun listReturnsEmptyEmbeddedOn404() = runBlocking {
51              repeat(2) {
52                  server.enqueue(MockResponse().setResponseCode(404))
53              }
54              val result = repository.list("disk:/missing", limit = 10, offset = 0)
55              assertTrue(result.embedded?.items?.isEmpty() == true)
56          }
57
58          @Test
59          fun diskInfoThrowsAuthExceptionOn401() = runBlocking {
60              server.enqueue(MockResponse().setResponseCode(401))
61              try {
62                  repository.diskInfo()
63                  error("Expected YandexDiskAuthException")
64              } catch (_: YandexDiskAuthException) {
65                  // expected
66              }
67          }
68      }
69
```

Автотест domain-vault/src/test/java/com/github/nullptroma/wallenc/domain/vault/storages/common/StorageSyncJournalBufferTest.kt

```
1   package com.github.nullptroma.wallenc.domain.vault.storages.common
```

```
2
3       import com.github.nullptroma.wallenc.domain.datatypes.StorageSyncJournalEntry
4       import com.github.nullptroma.wallenc.domain.datatypes.StorageSyncOperation
5       import com.github.nullptroma.wallenc.domain.datatypes.StorageSyncRevision
6       import kotlinx.coroutines.runBlocking
7       import org.junit.Assert.assertEquals
8       import org.junit.Assert.assertTrue
9       import org.junit.Test
10      import java.time.Instant
11      import java.util.concurrent.atomic.AtomicReference
12
13      class StorageSyncJournalBufferTest {
14
15          @Test
16          fun flushRestoresPendingOnWriteFailure() = runBlocking {
17              val disk = AtomicReference(emptyMap<String, StorageSyncJournalEntry>())
18              var shouldFail = true
19              val buffer = StorageSyncJournalBuffer(
20                  syncActorId = "actor",
21                  originStorageUuid = null,
22                  readJournal = { disk.get() },
23                  writeJournal = {
24                      if (shouldFail) {
25                          error("disk unavailable")
26                      }
27                      disk.set(it)
28                  },
29              )
30              val entry = buffer.buildEntry("/a.txt", StorageSyncOperation.UPSERT, 1L)
31              try {
32                  buffer.appendEntry("/a.txt", entry)
33              } catch (_: IllegalStateException) {
34                  // expected
35              }
36              shouldFail = false
37              buffer.flushPending()
38              assertTrue(disk.get().containsKey("/a.txt"))
39              assertEquals(1L, disk.get()["/a.txt"]?.revision?.sequence)
40          }
41      }
42
```

1.5 Модуль :usecases

Автотест usecases/src/test/java/com/github/nullptroma/wallenc/usecases/StorageDomainUseCasesTest.kt

```kotlin
package com.github.nullptroma.wallenc.usecases

import com.github.nullptroma.wallenc.domain.datatypes.TextSecretEntryRecord
import com.github.nullptroma.wallenc.usecases.fakes.FakeStorage
import kotlinx.coroutines.flow.first
import kotlinx.coroutines.test.runTest
import org.junit.Assert.assertEquals
import org.junit.Assert.assertNotNull
import org.junit.Assert.assertNull
import org.junit.Assert.assertTrue
import org.junit.Test

class StorageDomainUseCasesTest {

    @Test
    fun twoFaCrudWorksAndPersists() = runTest {
        val storage = FakeStorage()
        val useCase = ManageTwoFaTokensUseCase()

        val created = useCase.create(
            storageInfo = storage,
            issuer = "GitHub",
            account = "user@example.com",
            secret = "SECRET",
            notes = "primary",
        )
        assertNotNull(created.id)
        assertEquals(1, useCase.observe(storage).first().size)

        val updated = useCase.update(
            storageInfo = storage,
            token = created.copy(issuer = "GitHubUpdated"),
        )
        assertTrue(updated)
        assertEquals("GitHubUpdated", useCase.get(storage, created.id)?.issuer)

        val removed = useCase.delete(storage, created.id)
        assertTrue(removed)
        assertTrue(useCase.observe(storage).first().isEmpty())
    }

    @Test
    fun twoFaInvalidJsonFallsBackToEmptyList() = runTest {
```

```
            val storage = FakeStorage().apply {
                setDomainFile(StorageDomainDataFiles.TWO_FA_TOKENS_FILE, "not-json")
            }
            val useCase = ManageTwoFaTokensUseCase()
            assertTrue(useCase.observe(storage).first().isEmpty())
        }

        @Test
        fun textSecretsCrudWorksWithOptionalLabels() = runTest {
            val storage = FakeStorage()
            val useCase = ManageTextSecretsUseCase()

            val created = useCase.create(
                storageInfo = storage,
                title = "Server Credentials",
                items = listOf(
                    TextSecretEntryRecord(label = "username", value = "admin"),
                    TextSecretEntryRecord(label = null, value = "password"),
                ),
            )
            assertEquals(1, useCase.observe(storage).first().size)

            val updated = useCase.update(
                storageInfo = storage,
                secret = created.copy(
                    title = "Prod Credentials",
                    items = created.items + TextSecretEntryRecord(label = "url", value = "https://x.dev"),
                ),
            )
            assertTrue(updated)
            val loaded = useCase.get(storage, created.id)
            assertEquals("Prod Credentials", loaded?.title)
            assertEquals(3, loaded?.items?.size)

            val deleted = useCase.delete(storage, created.id)
            assertTrue(deleted)
            assertNull(useCase.get(storage, created.id))
        }

        @Test
        fun textSecretsInvalidJsonFallsBackToEmptyList() = runTest {
            val storage = FakeStorage().apply {
                setDomainFile(StorageDomainDataFiles.TEXT_SECRETS_FILE, "{broken")
            }
```

```
88              val useCase = ManageTextSecretsUseCase()
89              assertTrue(useCase.observe(storage).first().isEmpty())
90          }
91      }
92
```

Автотест usecases/src/test/java/com/github/nullptroma/wallenc/usecases/StorageSyncEncryptionCompatTest.kt

```
1   package com.github.nullptroma.wallenc.usecases
2
3   import com.github.nullptroma.wallenc.domain.datatypes.StorageEncryptionInfo
4   import com.github.nullptroma.wallenc.domain.interfaces.StorageSyncGroup
5   import com.github.nullptroma.wallenc.domain.interfaces.StorageSyncGroupEncryptionKind
6   import com.github.nullptroma.wallenc.usecases.fakes.FakeMetaInfo
7   import com.github.nullptroma.wallenc.usecases.fakes.FakeStorage
8   import org.junit.Assert.assertFalse
9   import org.junit.Assert.assertTrue
10  import org.junit.Test
11  import java.util.UUID
12
13  class StorageSyncEncryptionCompatTest {
14
15      @Test
16      fun storageWithoutEncInfoIsCompatible() {
17          val storage = FakeStorage(uuid = UUID.randomUUID(), meta = FakeMetaInfo(encInfo = null))
18          val group = StorageSyncGroup(
19              id = "g1",
20              storageUuids = setOf(storage.uuid),
21              encryptionKind = StorageSyncGroupEncryptionKind.NONE,
22          )
23          assertTrue(isStorageCompatibleWithGroup(storage = storage, group = group))
24      }
25
26      @Test
27      fun storageWithEncInfoIsIncompatible() {
28          val storage = FakeStorage(
29              uuid = UUID.randomUUID(),
30              meta = FakeMetaInfo(
31                  encInfo = StorageEncryptionInfo(encryptedTestData = "x", pathIv = ByteArray(16)),
32              ),
33          )
34          val group = StorageSyncGroup(
35              id = "g1",
36              storageUuids = setOf(storage.uuid),
```

```
                encryptionKind = StorageSyncGroupEncryptionKind.NONE,
            )
            assertFalse(isStorageCompatibleWithGroup(storage = storage, group = group))
        }
    }

```

Автотест usecases/src/test/java/com/github/nullptroma/wallenc/usecases/StorageSyncEngineTest.kt

```
package com.github.nullptroma.wallenc.usecases

import com.github.nullptroma.wallenc.domain.datatypes.StorageSyncJournalEntry
import com.github.nullptroma.wallenc.domain.datatypes.StorageSyncOperation
import com.github.nullptroma.wallenc.domain.datatypes.StorageSyncPaths
import com.github.nullptroma.wallenc.domain.datatypes.StorageSyncRevision
import com.github.nullptroma.wallenc.domain.interfaces.StorageSyncGroup
import com.github.nullptroma.wallenc.domain.interfaces.StorageSyncGroupEncryptionKind
import com.github.nullptroma.wallenc.domain.tasks.TaskProgressLabel
import com.github.nullptroma.wallenc.usecases.fakes.FakeStorage
import com.github.nullptroma.wallenc.usecases.fakes.FakeStorageAccessor
import com.github.nullptroma.wallenc.usecases.fakes.FakeStorageSyncGroupStore
import com.github.nullptroma.wallenc.usecases.fakes.FakeVaultsManager
import kotlinx.coroutines.CancellationException
import kotlinx.coroutines.async
import kotlinx.coroutines.runBlocking
import org.junit.Assert.assertArrayEquals
import org.junit.Assert.assertEquals
import org.junit.Assert.assertNull
import org.junit.Assert.assertTrue
import org.junit.Test
import java.time.Instant

class StorageSyncEngineTest {

    private fun norm(path: String): String = StorageSyncPaths.normalize(path)

    @Test
    fun syncAllGroupsReportsNoGroupsWhenEmpty() = runBlocking {
        val labels = mutableListOf<TaskProgressLabel?>()
        val engine = createEngine(storages = emptyList(), groups = emptyList())
        engine.syncAllGroups { _, label -> labels.add(label) }
        assertTrue(labels.any { it is TaskProgressLabel.SyncNoGroups })
    }

    @Test
```

```
37          fun syncGroupCopiesFileFromSourceToTarget() = runBlocking {
38              val source = FakeStorage()
39              val target = FakeStorage()
40              val path = "shared/doc.txt"
41              val payload = "sync-payload".encodeToByteArray()
42              source.putFile(path, payload)
43
44              val entry = StorageSyncJournalEntry(
45                  path = path,
46                  operation = StorageSyncOperation.UPSERT,
47                  revision = StorageSyncRevision(
48                      sequence = 1L,
49                      actorId = "actor-a",
50                      createdAt = Instant.parse("2024-01-01T00:00:00Z"),
51                  ),
52                  size = payload.size.toLong(),
53                  originStorageUuid = source.uuid,
54              )
55              source.addSyncJournalEntry(entry)
56
57              val groupId = "group-1"
58              val group = StorageSyncGroup(
59                  id = groupId,
60                  storageUuids = setOf(source.uuid, target.uuid),
61                  encryptionKind = StorageSyncGroupEncryptionKind.NONE,
62              )
63              val labels = mutableListOf<TaskProgressLabel?>()
64              val engine = createEngine(
65                  storages = listOf(source, target),
66                  groups = listOf(group),
67              )
68              engine.syncGroup(groupId) { _, label -> labels.add(label) }
69
70              assertArrayEquals(payload, target.fileBytes(path))
71              assertTrue(labels.any { it is TaskProgressLabel.SyncGroupCompleted })
72              assertTrue(target.syncJournalEntries().any { it.path == norm(path) })
73          }
74
75          @Test
76          fun syncGroupSkippedWhenFewerThanTwoStorages() = runBlocking {
77              val only = FakeStorage()
78              val groupId = "solo"
79              val group = StorageSyncGroup(
80                  id = groupId,
81                  storageUuids = setOf(only.uuid),
```

```
                encryptionKind = StorageSyncGroupEncryptionKind.NONE,
            )
            val labels = mutableListOf<TaskProgressLabel?>()
            val engine = createEngine(storages = listOf(only), groups = listOf(group))
            engine.syncGroup(groupId) { _, label -> labels.add(label) }
            assertTrue(labels.any { it is TaskProgressLabel.SyncGroupSkippedTooFewStorages })
        }

        @Test
        fun syncGroupDeleteRemovesFileOnTarget() = runBlocking {
            val source = FakeStorage()
            val target = FakeStorage()
            val path = "obsolete.bin"
            target.putFile(path, "old".encodeToByteArray())

            val entry = StorageSyncJournalEntry(
                path = path,
                operation = StorageSyncOperation.DELETE,
                revision = StorageSyncRevision(
                    sequence = 2L,
                    actorId = "actor-b",
                    createdAt = Instant.parse("2024-06-01T00:00:00Z"),
                ),
            )
            source.addSyncJournalEntry(entry)

            val group = StorageSyncGroup(
                id = "delete-group",
                storageUuids = setOf(source.uuid, target.uuid),
                encryptionKind = StorageSyncGroupEncryptionKind.NONE,
            )
            val engine = createEngine(
                storages = listOf(source, target),
                groups = listOf(group),
            )
            engine.syncGroup(group.id) { _, _ -> }

            assertNull(target.fileBytes(path))
            val targetEntry = target.syncJournalEntries().single { it.path == norm(path) }
            assertEquals(2L, targetEntry.revision.sequence)
            assertEquals("actor-b", targetEntry.revision.actorId)
            assertEquals(StorageSyncOperation.DELETE, targetEntry.operation)
        }

        @Test
```

```
127         fun syncSkipsWhenTargetRevisionAlreadyWinner() = runBlocking {
128             val source = FakeStorage()
129             val target = FakeStorage()
130             val path = "already-synced.txt"
131             val payload = "same".encodeToByteArray()
132             source.putFile(path, payload)
133             target.putFile(path, payload)
134 
135             val winner = StorageSyncJournalEntry(
136                 path = path,
137                 operation = StorageSyncOperation.UPSERT,
138                 revision = StorageSyncRevision(
139                     sequence = 5L,
140                     actorId = "winner-actor",
141                     createdAt = Instant.parse("2024-08-01T00:00:00Z"),
142                 ),
143                 size = payload.size.toLong(),
144             )
145             source.addSyncJournalEntry(winner)
146             target.addSyncJournalEntry(winner)
147 
148             val group = StorageSyncGroup(
149                 id = "skip-group",
150                 storageUuids = setOf(source.uuid, target.uuid),
151                 encryptionKind = StorageSyncGroupEncryptionKind.NONE,
152             )
153             val engine = createEngine(
154                 storages = listOf(source, target),
155                 groups = listOf(group),
156             )
157             engine.syncGroup(group.id) { _, _ -> }
158 
159             val targetJournal = target.syncJournalEntries().single { it.path == norm(path) }
160             assertEquals(winner.revision, targetJournal.revision)
161             assertEquals(winner.operation, targetJournal.operation)
162         }
163 
164         @Test
165         fun openReadDoesNotChangeJournal() = runBlocking {
166             val storage = FakeStorage()
167             val path = "read-only.txt"
168             storage.putFile(path, "data".encodeToByteArray())
169             val before = storage.syncJournalEntries().size
170 
171             storage.accessor.openRead(path).use { it.readBytes() }
```

```
172
173             assertEquals(before, storage.syncJournalEntries().size)
174         }
175
176         @Test
177         fun deleteWithRecordSyncJournalFalseDoesNotBumpSequence() = runBlocking {
178             val storage = FakeStorage()
179             val path = "to-delete.txt"
180             storage.putFile(path, "x".encodeToByteArray())
181             storage.addSyncJournalEntry(
182                 StorageSyncJournalEntry(
183                     path = path,
184                     operation = StorageSyncOperation.UPSERT,
185                     revision = StorageSyncRevision(10L, "prior", Instant.EPOCH),
186                 ),
187             )
188
189             storage.accessor.delete(path, recordSyncJournal = false)
190
191             assertNull(storage.fileBytes(path))
192             val entry = storage.syncJournalEntries().single { it.path == norm(path) }
193             assertEquals(10L, entry.revision.sequence)
194             assertEquals(StorageSyncOperation.UPSERT, entry.operation)
195         }
196
197         @Test
198         fun syncGroupTrashSoftDeletesOnTarget() = runBlocking {
199             val source = FakeStorage()
200             val target = FakeStorage()
201             val path = "trashed/doc.txt"
202             val payload = "keep-in-trash".encodeToByteArray()
203             source.putFile(path, payload)
204             target.putFile(path, payload)
205
206             val entry = StorageSyncJournalEntry(
207                 path = path,
208                 operation = StorageSyncOperation.TRASH,
209                 revision = StorageSyncRevision(
210                     sequence = 3L,
211                     actorId = "actor-trash",
212                     createdAt = Instant.parse("2024-07-01T00:00:00Z"),
213                 ),
214             )
215             source.addSyncJournalEntry(entry)
216
```

```
            val group = StorageSyncGroup(
                id = "trash-group",
                storageUuids = setOf(source.uuid, target.uuid),
                encryptionKind = StorageSyncGroupEncryptionKind.NONE,
            )
            val engine = createEngine(
                storages = listOf(source, target),
                groups = listOf(group),
            )
            engine.syncGroup(group.id) { _, _ -> }

            assertArrayEquals(payload, target.fileBytes(path))
            assertTrue(norm(path) in (target.accessor as FakeStorageAccessor).trashedPaths)
            val targetEntry = target.syncJournalEntries().single { it.path == norm(path) }
            assertEquals(3L, targetEntry.revision.sequence)
            assertEquals("actor-trash", targetEntry.revision.actorId)
            assertEquals(StorageSyncOperation.TRASH, targetEntry.operation)
        }

        @Test
        fun syncGroupStopsWhenLockCannotBeAcquired() = runBlocking {
            val first = FakeStorage()
            val second = FakeStorage().apply { setAcquireLockResult(false) }
            val group = StorageSyncGroup(
                id = "lock-fail",
                storageUuids = setOf(first.uuid, second.uuid),
                encryptionKind = StorageSyncGroupEncryptionKind.NONE,
            )
            first.addSyncJournalEntry(
                StorageSyncJournalEntry(
                    path = "a.txt",
                    operation = StorageSyncOperation.UPSERT,
                    revision = StorageSyncRevision(1L, "x", Instant.EPOCH),
                ),
            )
            val labels = mutableListOf<TaskProgressLabel?>()
            val engine = createEngine(
                storages = listOf(first, second),
                groups = listOf(group),
            )
            engine.syncGroup(group.id) { _, label -> labels.add(label) }
            assertTrue(labels.any { it is TaskProgressLabel.SyncGroupLockFailed })
            assertNull((first.accessor as FakeStorageAccessor).syncLock)
            assertNull((second.accessor as FakeStorageAccessor).syncLock)
        }
```

```
262
263         @Test
264         fun syncGroupReleasesLocksAfterSuccessfulSync() = runBlocking {
265             val source = FakeStorage()
266             val target = FakeStorage()
267             source.addSyncJournalEntry(
268                 StorageSyncJournalEntry(
269                     path = "a.txt",
270                     operation = StorageSyncOperation.UPSERT,
271                     revision = StorageSyncRevision(1L, "x", Instant.EPOCH),
272                 ),
273             )
274             source.putFile("a.txt", "x".encodeToByteArray())
275             val group = StorageSyncGroup(
276                 id = "ok",
277                 storageUuids = setOf(source.uuid, target.uuid),
278                 encryptionKind = StorageSyncGroupEncryptionKind.NONE,
279             )
280             val engine = createEngine(
281                 storages = listOf(source, target),
282                 groups = listOf(group),
283             )
284             engine.syncGroup(group.id) { _, _ -> }
285             assertNull((source.accessor as FakeStorageAccessor).syncLock)
286             assertNull((target.accessor as FakeStorageAccessor).syncLock)
287         }
288
289         @Test
290         fun syncGroupReleasesLocksWhenJournalReadFails() = runBlocking {
291             val first = FakeStorage()
292             val second = FakeStorage()
293             (first.accessor as FakeStorageAccessor).readSyncJournalThrows =
294                 IllegalStateException("journal read failed")
295             val group = StorageSyncGroup(
296                 id = "journal-fail",
297                 storageUuids = setOf(first.uuid, second.uuid),
298                 encryptionKind = StorageSyncGroupEncryptionKind.NONE,
299             )
300             val engine = createEngine(
301                 storages = listOf(first, second),
302                 groups = listOf(group),
303             )
304             runCatching { engine.syncGroup(group.id) { _, _ -> } }
305             assertNull((first.accessor as FakeStorageAccessor).syncLock)
306             assertNull((second.accessor as FakeStorageAccessor).syncLock)
```

```
307         }
308
309         @Test
310         fun syncGroupCooperativeCancellationReleasesLocks() = runBlocking {
311             val source = FakeStorage()
312             val target = FakeStorage()
313             val path = "slow.txt"
314             val payload = "payload".encodeToByteArray()
315             source.putFile(path, payload)
316             (source.accessor as FakeStorageAccessor).openReadDelayMs = 5_000
317             source.addSyncJournalEntry(
318                 StorageSyncJournalEntry(
319                     path = path,
320                     operation = StorageSyncOperation.UPSERT,
321                     revision = StorageSyncRevision(1L, "actor", Instant.EPOCH),
322                     size = payload.size.toLong(),
323                 ),
324             )
325             val group = StorageSyncGroup(
326                 id = "cancel-group",
327                 storageUuids = setOf(source.uuid, target.uuid),
328                 encryptionKind = StorageSyncGroupEncryptionKind.NONE,
329             )
330             val engine = createEngine(
331                 storages = listOf(source, target),
332                 groups = listOf(group),
333             )
334             val job = async {
335                 engine.syncGroup(group.id) { _, _ -> }
336             }
337             kotlinx.coroutines.delay(50)
338             job.cancel()
339             try {
340                 job.await()
341             } catch (_: CancellationException) {
342                 // expected
343             }
344             assertNull((source.accessor as FakeStorageAccessor).syncLock)
345             assertNull((target.accessor as FakeStorageAccessor).syncLock)
346         }
347
348         @Test
349         fun syncGroupReleasesLocksWhenJournalEmpty() = runBlocking {
350             val first = FakeStorage()
351             val second = FakeStorage()
```

```
            val group = StorageSyncGroup(
                id = "empty-journal",
                storageUuids = setOf(first.uuid, second.uuid),
                encryptionKind = StorageSyncGroupEncryptionKind.NONE,
            )
            val labels = mutableListOf<TaskProgressLabel?>()
            val engine = createEngine(
                storages = listOf(first, second),
                groups = listOf(group),
            )
            engine.syncGroup(group.id) { _, label -> labels.add(label) }
            assertTrue(labels.any { it is TaskProgressLabel.SyncGroupNoJournalEntries })
            assertNull((first.accessor as FakeStorageAccessor).syncLock)
            assertNull((second.accessor as FakeStorageAccessor).syncLock)
        }

        private fun createEngine(
            storages: List<FakeStorage>,
            groups: List<StorageSyncGroup>,
        ): StorageSyncEngine {
            val vaultsManager = FakeVaultsManager(storages)
            val findStorage = FindStorageUseCase(vaultsManager)
            return StorageSyncEngine(
                vaultsManager = vaultsManager,
                groupStore = FakeStorageSyncGroupStore(groups),
                findStorageUseCase = findStorage,
            )
        }
    }

```

Автотест usecases/src/test/java/com/github/nullptroma/wallenc/usecases/StorageSyncJournalMergeTest.kt

```
package com.github.nullptroma.wallenc.usecases

import com.github.nullptroma.wallenc.domain.datatypes.StorageSyncJournalEntry
import com.github.nullptroma.wallenc.domain.datatypes.StorageSyncJournalMerge
import com.github.nullptroma.wallenc.domain.datatypes.StorageSyncOperation
import com.github.nullptroma.wallenc.domain.datatypes.StorageSyncPaths
import com.github.nullptroma.wallenc.domain.datatypes.StorageSyncRevision
import org.junit.Assert.assertEquals
import org.junit.Assert.assertFalse
import org.junit.Assert.assertTrue
import org.junit.Test
import java.time.Instant
```

```
class StorageSyncJournalMergeTest {

    @Test
    fun mergeKeepsSingleEntryPerPath() {
        val older = entry(path = "/a.txt", sequence = 1L)
        val newer = entry(path = "/a.txt", sequence = 2L)
        val merged = StorageSyncJournalMerge.merge(
            mapOf("/a.txt" to older),
            mapOf("/a.txt" to newer),
        )
        assertEquals(1, merged.size)
        assertEquals(2L, merged["/a.txt"]?.revision?.sequence)
    }

    @Test
    fun isSyncableUserPathExcludesEncDirAndJournal() {
        assertFalse(StorageSyncPaths.isSyncableUserPath("/88a048ed-enc-dir/sync-journal.json"))
        assertFalse(StorageSyncPaths.isSyncableUserPath("/wallenc-yandex-system/sync-journal.json"))
        assertTrue(StorageSyncPaths.isSyncableUserPath("/wallenc-data/text-secrets.json"))
    }

    private fun entry(path: String, sequence: Long): StorageSyncJournalEntry =
        StorageSyncJournalEntry(
            path = path,
            operation = StorageSyncOperation.UPSERT,
            revision = StorageSyncRevision(
                sequence = sequence,
                actorId = "actor",
                createdAt = Instant.EPOCH,
            ),
        )
}
```

Автотест usecases/src/test/java/com/github/nullptroma/wallenc/usecases/TwoFaTotpTest.kt

```
package com.github.nullptroma.wallenc.usecases

import com.github.nullptroma.wallenc.domain.datatypes.TwoFaTokenRecord
import com.eatthepath.otp.TimeBasedOneTimePasswordGenerator
import org.junit.Assert.assertEquals
import org.junit.Assert.assertNotNull
import org.junit.Assert.assertTrue
```

```
8       import org.junit.Test
9       import java.time.Duration
10      import java.time.Instant
11      import javax.crypto.spec.SecretKeySpec
12
13      class TwoFaTotpTest {
14
15          @Test
16          fun buildTwoFaCodeStateMatchesJavaOtpForKnownSecret() {
17              val secretBase32 = "JBSWY3DPEHPK3PXP"
18              val token = TwoFaTokenRecord(
19                  id = "1",
20                  issuer = "Test",
21                  account = "user",
22                  secret = secretBase32,
23                  digits = 6,
24                  periodSeconds = 30,
25                  algorithm = "SHA1",
26                  notes = null,
27              )
28              val nowMillis = 1_700_000_000_000L
29              val state = buildTwoFaCodeState(token, nowMillis)
30              assertNotNull(state)
31              val expected = generateReferenceTotp(secretBase32, nowMillis, 6, 30, "HmacSHA1")
32              assertEquals(expected, state!!.code)
33              assertTrue(state.secondsUntilRefresh in 1..30)
34          }
35
36          @Test
37          fun totpPeriodProgressIsContinuousWithinPeriod() {
38              val period = 30
39              val periodStartMillis = 1_700_000_100_000L // epoch sec кратна period
40              assertEquals(0f, totpPeriodProgress(periodStartMillis, period), 0.001f)
41              assertEquals(0.5f, totpPeriodProgress(periodStartMillis + 15_000L, period), 0.001f)
42              assertEquals(29f / 30f, totpPeriodProgress(periodStartMillis + 29_000L, period),
    0.001f)
43          }
44
45          @Test
46          fun totpSecondsUntilRefreshCountsDownWithinPeriod() {
47              val period = 30
48              val t0 = 1_700_000_100_000L
49              assertEquals(30, totpSecondsUntilRefresh(t0, period))
50              assertEquals(15, totpSecondsUntilRefresh(t0 + 15_000L, period))
51              assertEquals(1, totpSecondsUntilRefresh(t0 + 29_000L, period))
```

```
52          }
53
54          @Test
55          fun buildTwoFaCodeStateReturnsNullForInvalidSecret() {
56              val token = TwoFaTokenRecord(
57                  id = "1",
58                  issuer = "Test",
59                  account = "user",
60                  secret = "!!!not-base32!!!",
61                  digits = 6,
62                  periodSeconds = 30,
63                  algorithm = "SHA1",
64                  notes = null,
65              )
66              assertEquals(null, buildTwoFaCodeState(token, System.currentTimeMillis()))
67          }
68
69          private fun generateReferenceTotp(
70              secretBase32: String,
71              nowMillis: Long,
72              digits: Int,
73              periodSeconds: Int,
74              macAlgorithm: String,
75          ): String {
76              val key = decodeBase32(secretBase32)
77              val generator = TimeBasedOneTimePasswordGenerator(
78                  Duration.ofSeconds(periodSeconds.toLong()),
79                  digits,
80                  macAlgorithm,
81              )
82              val otp = generator.generateOneTimePassword(
83                  SecretKeySpec(key, "RAW"),
84                  Instant.ofEpochMilli(nowMillis),
85              )
86              return otp.toString().padStart(digits, '0')
87          }
88
89          private fun decodeBase32(input: String): ByteArray {
90              val alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ234567"
91              val clean = input.uppercase().replace(" ", "").replace("=", "")
92              var buffer = 0
93              var bitsLeft = 0
94              val out = ArrayList<Byte>()
95              for (ch in clean) {
96                  val value = alphabet.indexOf(ch)
```

```
97                  buffer = (buffer shl 5) or value
98                  bitsLeft += 5
99                  if (bitsLeft >= 8) {
100                     out.add(((buffer shr (bitsLeft - 8)) and 0xFF).toByte())
101                     bitsLeft -= 8
102                 }
103             }
104             return out.toByteArray()
105         }
106     }
107
```

1.6 Модуль :ui

Автотест ui/src/androidTest/java/com/github/nullptroma/wallenc/ui/screens/main/screens/storage/secrets/TextSecretsScreenContentTest.kt

```kotlin
package com.github.nullptroma.wallenc.ui.screens.main.screens.storage.secrets

import androidx.compose.material3.MaterialTheme
import androidx.compose.ui.test.assertIsDisplayed
import androidx.compose.ui.test.junit4.createComposeRule
import androidx.compose.ui.test.onNodeWithText
import androidx.test.ext.junit.runners.AndroidJUnit4
import androidx.test.platform.app.InstrumentationRegistry
import com.github.nullptroma.wallenc.domain.datatypes.TextSecretEntryRecord
import com.github.nullptroma.wallenc.domain.datatypes.TextSecretRecord
import com.github.nullptroma.wallenc.ui.R
import org.junit.Rule
import org.junit.Test
import org.junit.runner.RunWith

@RunWith(AndroidJUnit4::class)
class TextSecretsScreenContentTest {

    @get:Rule
    val composeRule = createComposeRule()

    private val context get() = InstrumentationRegistry.getInstrumentation().targetContext

    @Test
    fun showsEmptyStateWhenNoSecrets() {
        composeRule.setContent {
            MaterialTheme {
                TextSecretsScreenContent(
                    uiState = TextSecretsScreenState(
                        isLoading = false,
                        isAvailable = true,
                        items = emptyList(),
                    ),
                )
            }
        }
composeRule.onNodeWithText(context.getString(R.string.text_secret_empty_state)).assertIsDispl
    }

    @Test
    fun showsSecretTitle() {
        val secret = TextSecretRecord(
```

```
                id = "s1",
                title = "Production DB",
                items = listOf(TextSecretEntryRecord(label = "user", value = "admin")),
            )
            composeRule.setContent {
                MaterialTheme {
                    TextSecretsScreenContent(
                        uiState = TextSecretsScreenState(
                            isLoading = false,
                            isAvailable = true,
                            items = listOf(secret),
                        ),
                    ) {
                        TextSecretListCard(
                            secret = secret,
                            onClick = {},
                            enabled = true,
                        )
                    }
                }
            }
            composeRule.onNodeWithText("Production DB").assertIsDisplayed()
        }
    }

```

Автотест ui/src/androidTest/java/com/github/nullptroma/wallenc/ui/screens/main/screens/storage/twofa/TwoFaTokensScreenContentTest.kt

```
package com.github.nullptroma.wallenc.ui.screens.main.screens.storage.twofa

import androidx.compose.foundation.layout.padding
import androidx.compose.material3.MaterialTheme
import androidx.compose.ui.Modifier
import androidx.compose.ui.test.junit4.createComposeRule
import androidx.compose.ui.test.onNodeWithText
import androidx.compose.ui.test.assertIsDisplayed
import androidx.compose.ui.unit.dp
import androidx.test.ext.junit.runners.AndroidJUnit4
import androidx.test.platform.app.InstrumentationRegistry
import com.github.nullptroma.wallenc.domain.datatypes.TwoFaTokenRecord
import com.github.nullptroma.wallenc.ui.R
import org.junit.Rule
import org.junit.Test
import org.junit.runner.RunWith

```

```
18      @RunWith(AndroidJUnit4::class)
19      class TwoFaTokensScreenContentTest {
20
21          @get:Rule
22          val composeRule = createComposeRule()
23
24          private val context get() = InstrumentationRegistry.getInstrumentation().targetContext
25
26          @Test
27          fun showsEmptyStateWhenNoTokens() {
28              composeRule.setContent {
29                  MaterialTheme {
30                      TwoFaTokensScreenContent(
31                          uiState = TwoFaTokensScreenState(
32                              isLoading = false,
33                              isAvailable = true,
34                              items = emptyList(),
35                          ),
36                      )
37                  }
38              }
39      composeRule.onNodeWithText(context.getString(R.string.two_fa_empty_state)).assertIsDisplayed(
40          }
41
42          @Test
43          fun showsIssuerAndAccountForToken() {
44              val token = TwoFaTokenRecord(
45                  id = "1",
46                  issuer = "GitHub",
47                  account = "user@example.com",
48                  secret = "SECRET",
49              )
50              composeRule.setContent {
51                  MaterialTheme {
52                      TwoFaTokensScreenContent(
53                          uiState = TwoFaTokensScreenState(
54                              isLoading = false,
55                              isAvailable = true,
56                              items = listOf(token),
57                          ),
58                      ) {
59                          TwoFaTokenListHeader(
60                              issuer = token.issuer,
61                              account = token.account,
62                              modifier = Modifier.padding(8.dp),
```

```
                        )
                    }
                }
            }
            composeRule.onNodeWithText("GitHub").assertIsDisplayed()
            composeRule.onNodeWithText("user@example.com").assertIsDisplayed()
        }
    }

```

Автотест ui/src/test/java/com/github/nullptroma/wallenc/ui/navigation/WallencDeepLinksTest.kt

```
package com.github.nullptroma.wallenc.ui.navigation

import android.content.Intent
import android.net.Uri
import org.junit.Assert.assertFalse
import org.junit.Assert.assertTrue
import org.junit.Test
import org.junit.runner.RunWith
import org.robolectric.RobolectricTestRunner
import org.robolectric.annotation.Config

@RunWith(RobolectricTestRunner::class)
@Config(sdk = [33])
class WallencDeepLinksTest {

    @Test
    fun matchesWallencViewIntent() {
        val intent = Intent(Intent.ACTION_VIEW,
Uri.parse(WallencDeepLinks.MAIN_URI_PATTERN))
        assertTrue(intent.matchesWallencDeepLink())
    }

    @Test
    fun rejectsUnrelatedIntent() {
        val intent = Intent(Intent.ACTION_MAIN)
        assertFalse(intent.matchesWallencDeepLink())
    }

    @Test
    fun matchesTasksAndSettingsHosts() {
        assertTrue(
            Intent(Intent.ACTION_VIEW, Uri.parse(WallencDeepLinks.TASKS_URI_PATTERN))
                .matchesWallencDeepLink(),
        )
```

```
            assertTrue(
                Intent(Intent.ACTION_VIEW, Uri.parse(WallencDeepLinks.SETTINGS_URI_PATTERN))
                    .matchesWallencDeepLink(),
            )
        }
    }

```

Автотест ui/src/test/java/com/github/nullptroma/wallenc/ui/resources/TaskProgressLabelsTest.kt

```
package com.github.nullptroma.wallenc.ui.resources

import com.github.nullptroma.wallenc.domain.tasks.TaskProgressLabel
import com.github.nullptroma.wallenc.domain.tasks.VaultTaskStep
import com.github.nullptroma.wallenc.ui.R
import org.junit.Assert.assertEquals
import org.junit.Test

class TaskProgressLabelsTest {

    private val resolver = object : UiStringResolver {
        override fun invoke(id: Int, vararg formatArgs: Any): String = "res:$id"

        override fun plurals(id: Int, quantity: Int, vararg formatArgs: Any): String = "plural:$id"
    }

    @Test
    fun syncNoGroups_mapsToStringRes() {
        val text = TaskProgressLabel.SyncNoGroups.resolve(resolver)
        assertEquals("res:${R.string.sync_progress_no_groups}", text)
    }

    @Test
    fun vaultTask_mapsToStringRes() {
        val text = TaskProgressLabel.VaultTask(VaultTaskStep.Save2FaToken).resolve(resolver)
        assertEquals("res:${R.string.task_progress_save_2fa_token}", text)
    }

    @Test
    fun clearContentProgress_mapsToStringRes() {
        val text = TaskProgressLabel.ClearContentProgress(3, 10).resolve(resolver)
        assertEquals("res:${R.string.task_progress_clear_content}", text)
    }
}
```

Автотест ui/src/test/java/com/github/nullptroma/wallenc/ui/resources/WallencUserNotificationMappingTest.kt

```
package com.github.nullptroma.wallenc.ui.resources

import com.github.nullptroma.wallenc.domain.errors.WallencException
import com.github.nullptroma.wallenc.ui.R
import org.junit.Assert.assertEquals
import org.junit.Test

class WallencUserNotificationMappingTest {

    @Test
    fun mapsFeatureStorageNotFound() {
        val notification = WallencException.Feature.StorageNotFound().toUserNotification()
        assertEquals(R.string.error_storage_not_found, notification.id)
    }

    @Test
    fun mapsStorageIncorrectKey() {
        val notification = WallencException.Storage.IncorrectKey().toUserNotification()
        assertEquals(R.string.error_incorrect_password, notification.id)
    }

    @Test
    fun mapsUnknown() {
        val notification = WallencException.Unknown(Exception("x")).toUserNotification()
        assertEquals(R.string.error_unknown, notification.id)
    }
}

```

Автотест ui/src/test/java/com/github/nullptroma/wallenc/ui/screens/main/screens/storage/StorageNavigationRoutesSmokeTest.kt

```
package com.github.nullptroma.wallenc.ui.screens.main.screens.storage

import com.github.nullptroma.wallenc.ui.screens.main.screens.storage.secrets.TextSecretDetailsRoute
import com.github.nullptroma.wallenc.ui.screens.main.screens.storage.secrets.TextSecretEditRoute
import com.github.nullptroma.wallenc.ui.screens.main.screens.storage.secrets.TextSecretsRoute
import com.github.nullptroma.wallenc.ui.screens.main.screens.storage.twofa.TwoFaTokensRoute
import org.junit.Assert.assertEquals
import org.junit.Assert.assertNull
```

```
import org.junit.Test

class StorageNavigationRoutesSmokeTest {

    @Test
    fun storageHomeRouteCarriesVaultAndStorageIds() {
        val route = StorageHomeRoute(
            vaultUuid = "vault-1",
            storageUuid = "storage-1",
        )
        assertEquals("vault-1", route.vaultUuid)
        assertEquals("storage-1", route.storageUuid)
    }

    @Test
    fun textSecretsRoutesCarryRequiredArguments() {
        val listRoute = TextSecretsRoute(storageUuid = "storage-1")
        val detailsRoute = TextSecretDetailsRoute(storageUuid = "storage-1", secretId = "secret-1")
        val editRoute = TextSecretEditRoute(storageUuid = "storage-1", secretId = null)
        val twoFaRoute = TwoFaTokensRoute(storageUuid = "storage-1")

        assertEquals("storage-1", listRoute.storageUuid)
        assertEquals("secret-1", detailsRoute.secretId)
        assertNull(editRoute.secretId)
        assertEquals("storage-1", twoFaRoute.storageUuid)
    }
}

```

Автотест ui/src/test/java/com/github/nullptroma/wallenc/ui/screens/main/screens/storage/twofa/OtpAuthUriParserTest.kt

```
package com.github.nullptroma.wallenc.ui.screens.main.screens.storage.twofa

import org.junit.Assert.assertEquals
import org.junit.Assert.assertNotNull
import org.junit.Assert.assertNull
import org.junit.Test
import org.junit.runner.RunWith
import org.robolectric.RobolectricTestRunner
import org.robolectric.annotation.Config

@RunWith(RobolectricTestRunner::class)
@Config(sdk = [33])
class OtpAuthUriParserTest {
```

```
14
15          @Test
16          fun parsesStandardTotpUri() {
17              val uri = "otpauth://totp/GitHub:user@example.com?secret=JBSWY3DPEHPK3PXP&issuer=
    GitHub&digits=6&period=30"
18              val parsed = parseOtpAuthTotpUri(uri)
19              assertNotNull(parsed)
20              assertEquals("GitHub", parsed!!.issuer)
21              assertEquals("user@example.com", parsed.account)
22              assertEquals("JBSWY3DPEHPK3PXP", parsed.secret)
23              assertEquals(6, parsed.digits)
24              assertEquals(30, parsed.periodSeconds)
25              assertEquals("SHA1", parsed.algorithm)
26          }
27
28          @Test
29          fun rejectsNonOtpauthScheme() {
30              assertNull(parseOtpAuthTotpUri("https://example.com"))
31          }
32
33          @Test
34          fun rejectsMissingSecret() {
35              assertNull(parseOtpAuthTotpUri("otpauth://totp/Test:account"))
36          }
37      }
38
```

Автотест ui/src/test/java/com/github/nullptroma/wallenc/ui/screens/main/screens/tasks/TaskPipelineViewModelTest.kt

```
1   package com.github.nullptroma.wallenc.ui.screens.main.screens.tasks
2
3   import com.github.nullptroma.wallenc.domain.tasks.TaskRunState
4   import com.github.nullptroma.wallenc.task.runtime.TaskOrchestrator
5   import com.github.nullptroma.wallenc.ui.resources.UiStringResolver
6   import io.mockk.every
7   import io.mockk.mockk
8   import kotlinx.coroutines.Dispatchers
9   import kotlinx.coroutines.delay
10  import kotlinx.coroutines.runBlocking
11  import kotlinx.coroutines.withTimeout
12  import org.junit.Assert.assertTrue
13  import org.junit.Test
14  import org.junit.runner.RunWith
15  import org.robolectric.RobolectricTestRunner
16  import org.robolectric.annotation.Config
17
```

```
18      @RunWith(RobolectricTestRunner::class)
19      @Config(sdk = [33])
20      class TaskPipelineViewModelTest {
21
22          @Test
23          fun startTestTaskEnqueuesWork() = runBlocking {
24              val orchestrator = TaskOrchestrator(Dispatchers.Default)
25              val uiStrings = mockk<UiStringResolver>()
26              every { uiStrings.invoke(any<Int>(), any()) } returns "Test task"
27              every { uiStrings.invoke(any<Int>()) } returns "Test"
28              every { uiStrings.plurals(any(), any(), any()) } returns "Plural"
29              val viewModel = TaskPipelineViewModel(orchestrator, uiStrings)
30
31              viewModel.startTestTask(durationSec = 0, infinityIndeterminateProgress = false)
32
33              withTimeout(5_000) {
34                  while (true) {
35                      val task = orchestrator.pipelineState.value.tasks.singleOrNull()
36                      if (task?.state is TaskRunState.Completed) break
37                      delay(25)
38                  }
39              }
40              val task = orchestrator.pipelineState.value.tasks.single()
41              assertTrue(task.state is TaskRunState.Completed)
42          }
43      }
44
```

1.7 Модуль :task-runtime

Автотест task-runtime/src/test/java/com/github/nullptroma/wallenc/task/runtime/TaskOrchestratorTest.kt

```kotlin
package com.github.nullptroma.wallenc.task.runtime

import com.github.nullptroma.wallenc.domain.errors.WallencException
import com.github.nullptroma.wallenc.domain.tasks.TaskLogLevel
import com.github.nullptroma.wallenc.domain.tasks.TaskProgressLabel
import com.github.nullptroma.wallenc.domain.tasks.TaskRunState
import kotlinx.coroutines.ExperimentalCoroutinesApi
import kotlinx.coroutines.test.StandardTestDispatcher
import kotlinx.coroutines.test.advanceTimeBy
import kotlinx.coroutines.test.advanceUntilIdle
import kotlinx.coroutines.test.runTest
import org.junit.Assert.assertTrue
import org.junit.Test

@OptIn(ExperimentalCoroutinesApi::class)
class TaskOrchestratorTest {

  private val dispatcher = StandardTestDispatcher()

  @Test
  fun enqueueCompletesTask() = runTest(dispatcher) {
    val orchestrator = TaskOrchestrator(dispatcher)
    val id = orchestrator.enqueue(
      title = "Test",
      dispatcher = dispatcher,
      work = { ctx ->
        ctx.reportProgress(0.5f, TaskProgressLabel.SyncPreparing(1))
      },
    )
    advanceUntilIdle()
    val task = orchestrator.pipelineState.value.tasks.first { it.id == id }
    assertTrue(task.state is TaskRunState.Completed)
  }

  @Test
  fun cancelAllMarksRunningTaskCancelled() = runTest(dispatcher) {
    val orchestrator = TaskOrchestrator(dispatcher)
    val id = orchestrator.enqueue(
      title = "Long",
      dispatcher = dispatcher,
      work = { ctx ->
        ctx.reportProgress(null, null)
        kotlinx.coroutines.delay(60_000)
```

```
            },
        )
        advanceTimeBy(1)
        orchestrator.cancelAll()
        advanceUntilIdle()
        val task = orchestrator.pipelineState.value.tasks.first { it.id == id }
        assertTrue(task.state is TaskRunState.Cancelled)
    }

    @Test
    fun cancelMarksTaskCancelled() = runTest(dispatcher) {
        val orchestrator = TaskOrchestrator(dispatcher)
        val id = orchestrator.enqueue(
            title = "Long",
            dispatcher = dispatcher,
            work = { ctx ->
                ctx.reportProgress(null, null)
                kotlinx.coroutines.delay(60_000)
            },
        )
        advanceTimeBy(1)
        orchestrator.cancel(id)
        advanceUntilIdle()
        val task = orchestrator.pipelineState.value.tasks.first { it.id == id }
        assertTrue(task.state is TaskRunState.Cancelled)
    }

    @Test
    fun failRecordsFailedState() = runTest(dispatcher) {
        val orchestrator = TaskOrchestrator(dispatcher)
        val id = orchestrator.enqueue(
            title = "Fail",
            dispatcher = dispatcher,
            work = { ctx ->
                ctx.fail(WallencException.Storage.FileNotFound())
            },
        )
        advanceUntilIdle()
        val task = orchestrator.pipelineState.value.tasks.first { it.id == id }
        val failed = task.state as TaskRunState.Failed
        assertTrue(failed.error is WallencException.Storage.FileNotFound)
    }

    @Test
    fun progressUpdatesRunningState() = runTest(dispatcher) {
```

```
89          val orchestrator = TaskOrchestrator(dispatcher)
90          val id = orchestrator.enqueue(
91            title = "Progress",
92            dispatcher = dispatcher,
93            work = { ctx ->
94              ctx.reportProgress(0.25f, TaskProgressLabel.SyncCompleted)
95            },
96          )
97          advanceUntilIdle()
98          val task = orchestrator.pipelineState.value.tasks.first { it.id == id }
99          assertTrue(task.state is TaskRunState.Completed)
100       }
101
102       @Test
103       fun logAppendsLine() = runTest(dispatcher) {
104         val orchestrator = TaskOrchestrator(dispatcher)
105         orchestrator.enqueue(
106           title = "Log",
107           dispatcher = dispatcher,
108           work = { ctx ->
109             ctx.log(TaskLogLevel.Info, "hello")
110           },
111         )
112         advanceUntilIdle()
113         assertTrue(orchestrator.logLines.value.any { it.message == "hello" })
114       }
115     }
116
```

1.8 Модуль :infrastructure-android

Автотест infrastructure-android/src/androidTest/java/com/github/nullptroma/wallenc/infrastructure/android/db/app/repository/YandexAccountRepositoryTest.kt

```kotlin
package com.github.nullptroma.wallenc.infrastructure.android.db.app.repository

import androidx.room.Room
import androidx.test.core.app.ApplicationProvider
import androidx.test.ext.junit.runners.AndroidJUnit4
import com.github.nullptroma.wallenc.domain.vault.model.YandexAccount
import com.github.nullptroma.wallenc.infrastructure.android.db.app.AppDb
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.runBlocking
import org.junit.After
import org.junit.Assert.assertEquals
import org.junit.Assert.assertNull
import org.junit.Before
import org.junit.Test
import org.junit.runner.RunWith

@RunWith(AndroidJUnit4::class)
class YandexAccountRepositoryTest {

    private lateinit var db: AppDb
    private lateinit var repository: YandexAccountRepository

    @Before
    fun setUp() {
        val context = ApplicationProvider.getApplicationContext<android.content.Context>()
        db = Room.inMemoryDatabaseBuilder(context, AppDb::class.java).build()
        repository = YandexAccountRepository(
            dao = db.yandexAccountDao,
            ioDispatcher = Dispatchers.IO,
        )
    }

    @After
    fun tearDown() {
        db.close()
    }

    @Test
    fun insertAndLoadByVaultUuid() = runBlocking {
        val account = YandexAccount(
            vaultUuid = "vault-1",
            yandexUserId = "user-1",
            email = "test@yandex.ru",
```

```
44                  oauthToken = "token-abc",
45              )
46              repository.insert(account)
47              val loaded = repository.getByVaultUuid("vault-1")
48              assertEquals(account, loaded)
49          }
50
51          @Test
52          fun updateCredentialsChangesToken() = runBlocking {
53              repository.insert(
54                  YandexAccount(
55                      vaultUuid = "vault-2",
56                      yandexUserId = "user-2",
57                      email = "old@yandex.ru",
58                      oauthToken = "old-token",
59                  ),
60              )
61              repository.updateCredentials("vault-2", "new@yandex.ru", "new-token")
62              val loaded = repository.getByVaultUuid("vault-2")
63              assertEquals("new@yandex.ru", loaded?.email)
64              assertEquals("new-token", loaded?.oauthToken)
65          }
66
67          @Test
68          fun deleteByVaultUuidRemovesRow() = runBlocking {
69              repository.insert(
70                  YandexAccount(
71                      vaultUuid = "vault-3",
72                      yandexUserId = "user-3",
73                      email = "x@yandex.ru",
74                      oauthToken = "t",
75                  ),
76              )
77              repository.deleteByVaultUuid("vault-3")
78              assertNull(repository.getByVaultUuid("vault-3"))
79          }
80      }
81
```

1.9 Модуль :app

Автотест app/src/androidTest/java/com/github/nullptroma/wallenc/app/integration/yandex/YandexDiskLiveIntegrationTest.kt

```kotlin
package com.github.nullptroma.wallenc.app.integration.yandex

import com.github.nullptroma.wallenc.domain.vault.network.yandexdisk.YandexDiskApiFactory
import com.github.nullptroma.wallenc.domain.vault.network.yandexdisk.repository.YandexDiskRepository
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.runBlocking
import org.junit.After
import org.junit.Assert.assertNotNull
import org.junit.Assert.assertTrue
import org.junit.Before
import org.junit.Test
import org.junit.runner.RunWith
import org.junit.runners.JUnit4

/**
 * Live-прогон Disk API. Пути только `app:/` — как в [YandexVault]; токен из Auth SDK
 * обычно имеет `cloud_api:disk.app_folder`, без полного доступа к `disk:/` (там 403).
 */
@RunWith(JUnit4::class)
class YandexDiskLiveIntegrationTest {

    private lateinit var repository: YandexDiskRepository
    private val testFolder = "app:/wallenc-integration-test"
    private val probeFileName = "wallenc-probe.txt"
    private val probePath = "$testFolder/$probeFileName"
    private val probePayload = "wallenc-integration-probe".encodeToByteArray()

    @Before
    fun setUp() {
        YandexTestCredentials.assumePresent()
        val token = YandexTestCredentials.oauthToken()!!
        repository = YandexDiskApiFactory.createRepositoryWithToken(token, Dispatchers.IO)
        runBlocking {
            runCatching { repository.createFolder(testFolder) }
            runCatching {
                repository.uploadBytes(probePath, probePayload, overwrite = true)
            }
        }
    }

    @After
    fun tearDown(): Unit = runBlocking {
```

```
            runCatching { repository.delete(probePath, permanently = true) }
        }

        @Test
        fun diskInfoReturnsQuota() = runBlocking {
            val info = repository.diskInfo()
            assertNotNull(info.totalSpace)
            assertTrue(info.totalSpace!! > 0)
        }

        @Test
        fun listTestFolderDoesNotThrow() = runBlocking {
            val result = repository.list(testFolder, limit = 10, offset = 0)
            assertNotNull(result)
        }

        @Test
        fun uploadAndDownloadRoundTrip() = runBlocking {
            val path = "$testFolder/roundtrip-${System.currentTimeMillis()}.bin"
            try {
                repository.uploadBytes(path, probePayload, overwrite = true)
                val downloaded = repository.openDownloadStream(path).use { it.readBytes() }
                assertTrue(downloaded.contentEquals(probePayload))
            } finally {
                runCatching { repository.delete(path, permanently = true) }
            }
        }
    }

```

Автотест app/src/androidTest/java/com/github/nullptroma/wallenc/app/integration/yandex/YandexTestCredentials.kt

```
package com.github.nullptroma.wallenc.app.integration.yandex

import androidx.test.platform.app.InstrumentationRegistry
import org.junit.Assume.assumeFalse

object YandexTestCredentials {
    fun oauthToken(): String? =
        InstrumentationRegistry.getArguments().getString("yandex.oauth.token")?.takeIf { it.isNotBlank() }

    fun assumePresent(message: String = "Добавьте yandex.test.oauth.token в local.properties") {
        assumeFalse(message, oauthToken().isNullOrBlank())
    }
```

```
13      }
14
```

1.9.1 Криптография и доменные ошибки

Класс `EncryptorTest` проверяет сценарии AES: `checkKey`, шифрование строк, байтовых массивов и потоков с верным и неверным ключом. `WallencExceptionMappingTest` покрывает преобразование файловых и сетевых исключений.

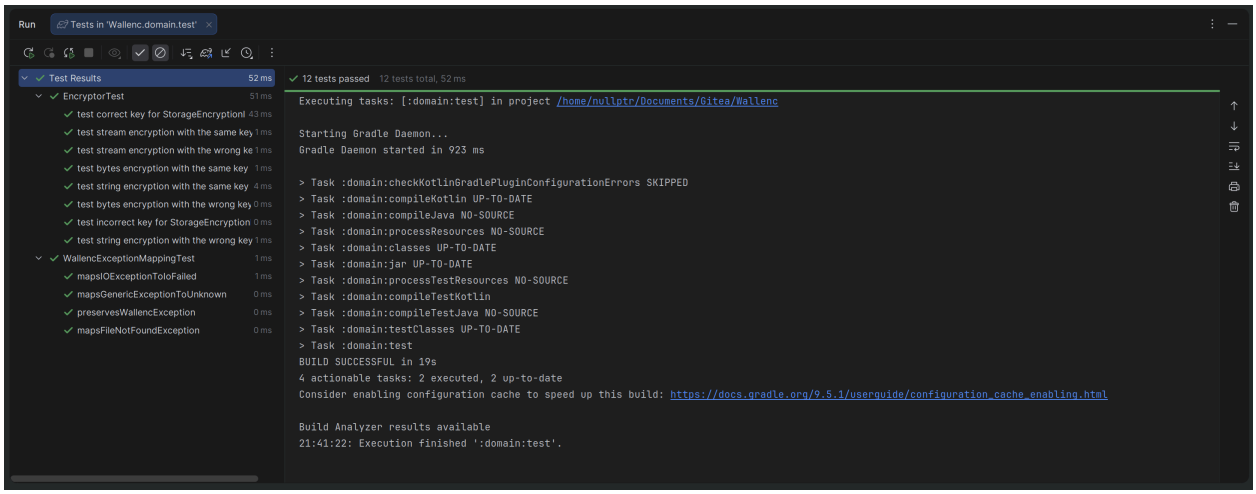

Рис. 1. Отчёт Gradle: модуль :domain, задача test

1.9.2 Синхронизация, 2FA и use cases

`StorageSyncEngineTest` моделирует группы синхронизации, копирование и удаление файлов, soft-delete, отмену и блокировки; отдельно проверяются слияние журнала и пропуск цели с актуальной ревизией. `TwoFaTotpTest` сверяет TOTP с эталоном Java OTP. `StorageDomainUseCasesTest` проверяет CRUD текстовых секретов и 2FA.

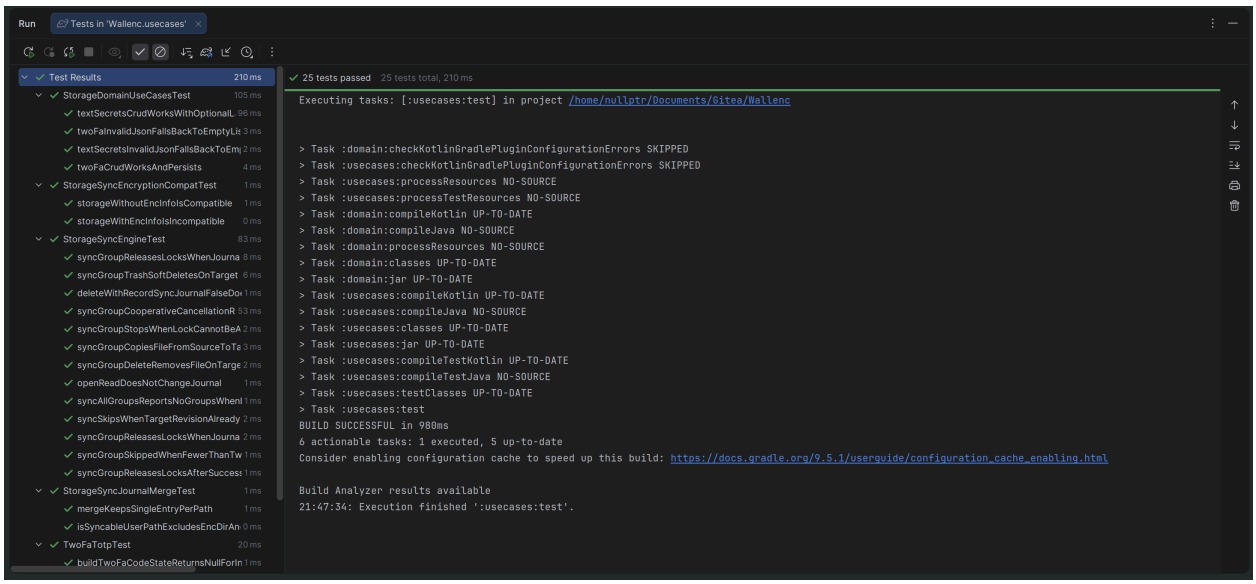

Рис. 2. Отчёт Gradle: модуль :usecases

1.9.3 Модуль :domain-vault

`YandexDiskRepositoryTest` использует мок HTTP: разбор `diskInfo`, пустой список при 404, `AuthException` при 401. `VaultThrowableMappingTest` покрывает сетевые и файловые ошибки vault.

1.9.4 Модуль :ui

Проверены чистые функции навигации, deep link, подписи уведомлений, парсинг OTP URI и постановка задачи в очередь (`TaskPipelineViewModelTest`).

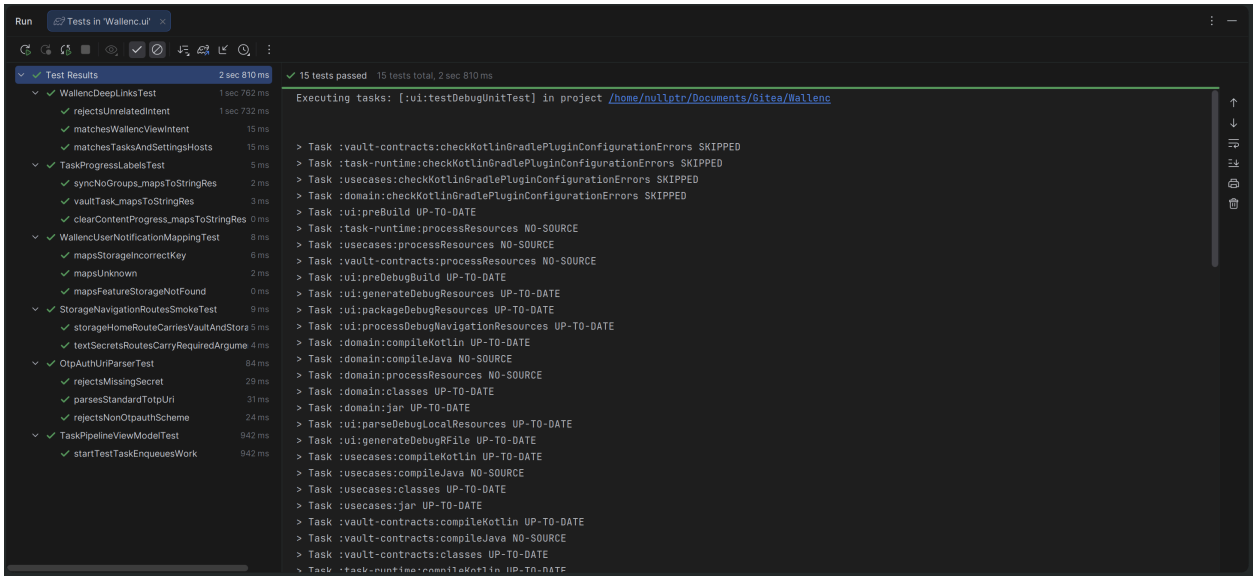

Рис. 3. Отчёт Gradle: модуль :ui

1.9.5 Модуль :task-runtime

`TaskOrchestratorTest` проверяет enqueue, progress, fail, cancel и cancelAll.

1.10 Инструментальные тесты (androidTest)

Запуск: `./gradlew connectedDebugAndroidTest`. Результат — на рис. Рис. 4.

Рис. 4. Gradle connectedDebugAndroidTest