

## **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

```
1 package com.github.nullptroma.wallenc.domain
2
3 import com.github.nullptroma.wallenc.domain.datatypes.EncryptKey
4 import com.github.nullptroma.wallenc.domain.encrypt.Encryptor
5 import junit.framework.TestCase.assertEquals
6 import junit.framework.TestCase.fail
7 import org.hamcrest.MatcherAssert.assertThat
8 import org.hamcrest.core.Is
9 import org.hamcrest.core.IsEqual
10 import org.hamcrest.core.IsNot
11 import org.junit.Assert.assertArrayEquals
12 import org.junit.Test
13 import java.io.ByteArrayInputStream
14 import java.io.ByteArrayOutputStream
15 import java.util.Random
16
17 class EncryptorTest {
18     val key1 = EncryptKey("key1")
19     val key2 = EncryptKey("key2")
20     val rnd = Random()
21
22     @Test
23     fun `test correct key for StorageEncryptionInfo`() {
24         val encInfo = Encryptor.generateEncryptionInfo(key1)
25         val res = Encryptor.checkKey(key = key1, encInfo = encInfo)
26         assertEquals(true, res)
27     }
28
29     @Test
30     fun `test incorrect key for StorageEncryptionInfo`() {
31         val encInfo = Encryptor.generateEncryptionInfo(key1)
32         val res = Encryptor.checkKey(key = key2, encInfo = encInfo)
33         assertEquals(false, res)
34     }
35
36     @Test
37     fun `test string encryption with the same key`() {
38         val text = "Hello world, my name is Wallenc!"
39         val encryptor = Encryptor(key1.toAesKey())
40         val encryptedText = encryptor.encryptString(text)
41
42         val newEncryptor = Encryptor(key1.toAesKey())
43         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        assertEquals(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")

```

```

135         }
136         catch (e: Exception) {
137             // good
138         }
139     }
140 }

```

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

```

1  package com.github.nullptroma.wallenc.domain.errors
2
3  import org.junit.Assert.assertEquals
4  import org.junit.Assert.assertTrue
5  import org.junit.Test
6  import java.io.FileNotFoundException
7  import java.io.IOException
8
9  class WallencExceptionMappingTest {
10
11     @Test
12     fun preservesWallencException() {
13         val original = WallencException.Feature.StorageNotFound()
14         assertEquals(original, original.toWallencException())
15     }
16
17     @Test
18     fun mapsFileNotFoundException() {
19         val mapped = FileNotFoundException("missing").toWallencException()
20         assertTrue(mapped is WallencException.Storage.FileNotFoundException)
21     }
22
23     @Test
24     fun mapsIOExceptionToIoFailed() {
25         val cause = IOException("disk")
26         val mapped = cause.toWallencException()
27         assertTrue(mapped is WallencException.Network.IoFailed)
28         assertEquals(cause, (mapped as WallencException.Network.IoFailed).cause)
29     }
30
31     @Test
32     fun mapsGenericExceptionToUnknown() {
33         val cause = Exception("boom")
34         val mapped = cause.toWallencException()
35         assertTrue(mapped is WallencException.Unknown)
36         assertEquals(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

```
1 package com.github.nullptroma.wallenc.domain.vault.errors
2
3 import com.github.nullptroma.wallenc.domain.errors.WallencException
4 import com.github.nullptroma.wallenc.domain.vault.network.yandexdisk.YandexDiskAuthException
5 import org.junit.Assert.assertEquals
6 import org.junit.Assert.assertTrue
7 import org.junit.Test
8 import okhttp3.ResponseBody.Companion.toResponseBody
9 import retrofit2.HttpException
10 import retrofit2.Response
11 import java.io.FileNotFoundException
12 import java.io.IOException
13 import java.net.SocketTimeoutException
14
15 class VaultThrowableMappingTest {
16
17     @Test
18     fun mapsYandexDiskAuthToAuthFailed() {
19         val mapped = YandexDiskAuthException("unauthorized").toVaultWallencException()
20         assertTrue(mapped is WallencException.Auth.Failed)
21     }
22
23     @Test
24     fun mapsHttpExceptionToNetworkHttpFailed() {
25         val response = Response.error<String>(401, "").toResponseBody(null)
26         val mapped = HttpException(response).toVaultWallencException()
27         assertTrue(mapped is WallencException.Network.HttpFailed)
28         assertEquals(401, (mapped as WallencException.Network.HttpFailed).statusCode)
29     }
30
31     @Test
32     fun mapsMissingOAuthTokenToTokenMissing() {
33         val mapped = IOException("Yandex OAuth token is missing").toVaultWallencException()
34         assertTrue(mapped is WallencException.Auth.TokenMissing)
35     }
36
37     @Test
38     fun mapsSocketTimeoutToOperationTimedOut() {
39         val mapped = SocketTimeoutException("timeout").toVaultWallencException()
40         assertTrue(mapped is WallencException.Network.OperationTimedOut)
41     }
42
43     @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())
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

```

ABTOTECT 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

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

```
1 package com.github.nullptroma.wallenc.usecases
2
3 import com.github.nullptroma.wallenc.domain.datatypes.TextSecretEntryRecord
4 import com.github.nullptroma.wallenc.usecases.fakes.FakeStorage
5 import kotlinx.coroutines.flow.first
6 import kotlinx.coroutines.test.runTest
7 import org.junit.Assert.assertEquals
8 import org.junit.Assert.assertNotNull
9 import org.junit.Assert.assertNull
10 import org.junit.Assert.assertTrue
11 import org.junit.Test
12
13 class StorageDomainUseCasesTest {
14
15     @Test
16     fun twoFaCrudWorksAndPersists() = runTest {
17         val storage = FakeStorage()
18         val useCase = ManageTwoFaTokensUseCase()
19
20         val created = useCase.create(
21             storageInfo = storage,
22             issuer = "GitHub",
23             account = "user@example.com",
24             secret = "SECRET",
25             notes = "primary",
26         )
27         assertNotNull(created.id)
28         assertEquals(1, useCase.observe(storage).first().size)
29
30         val updated = useCase.update(
31             storageInfo = storage,
32             token = created.copy(issuer = "GitHubUpdated"),
33         )
34         assertTrue(updated)
35         assertEquals("GitHubUpdated", useCase.get(storage, created.id)?.issuer)
36
37         val removed = useCase.delete(storage, created.id)
38         assertTrue(removed)
39         assertTrue(useCase.observe(storage).first().isEmpty())
40     }
41
42     @Test
43     fun twoFaInvalidJsonFallsBackToEmptyList() = runTest {
```

```

44         val storage = FakeStorage().apply {
45             setDomainFile(StorageDomainDataFiles.TWO_FA_TOKENS_FILE, "not-json")
46         }
47         val useCase = ManageTwoFaTokensUseCase()
48         assertTrue(useCase.observe(storage).first().isEmpty())
49     }
50
51     @Test
52     fun textSecretsCrudWorksOptionalLabels() = runTest {
53         val storage = FakeStorage()
54         val useCase = ManageTextSecretsUseCase()
55
56         val created = useCase.create(
57             storageInfo = storage,
58             title = "Server Credentials",
59             items = listOf(
60                 TextSecretEntryRecord(label = "username", value = "admin"),
61                 TextSecretEntryRecord(label = null, value = "password"),
62             ),
63         )
64         assertEquals(1, useCase.observe(storage).first().size)
65
66         val updated = useCase.update(
67             storageInfo = storage,
68             secret = created.copy(
69                 title = "Prod Credentials",
70                 items = created.items + TextSecretEntryRecord(label = "url", value =
71 "https://x.dev"),
72             ),
73         )
74         assertTrue(updated)
75         val loaded = useCase.get(storage, created.id)
76         assertEquals("Prod Credentials", loaded?.title)
77         assertEquals(3, loaded?.items?.size)
78
79         val deleted = useCase.delete(storage, created.id)
80         assertTrue(deleted)
81         assertNull(useCase.get(storage, created.id))
82     }
83
84     @Test
85     fun textSecretsInvalidJsonFallsBackToEmptyList() = runTest {
86         val storage = FakeStorage().apply {
87             setDomainFile(StorageDomainDataFiles.TEXT_SECRETS_FILE, "{broken}")
88         }

```

```

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

```

## ABTOTECT 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 =
18 null))
19         val group = StorageSyncGroup(
20             id = "g1",
21             storageUuids = setOf(storage.uuid),
22             encryptionKind = StorageSyncGroupEncryptionKind.NONE,
23         )
24         assertTrue(isStorageCompatibleWithGroup(storage = storage, group = group))
25     }
26
27     @Test
28     fun storageWithEncInfoIsIncompatible() {
29         val storage = FakeStorage(
30             uuid = UUID.randomUUID(),
31             meta = FakeMetaInfo(
32                 encInfo = StorageEncryptionInfo(encryptedTestData = "x", pathIv =
33 ByteArray(16)),
34             ),
35         )
36         val group = StorageSyncGroup(
37             id = "g1",
38             storageUuids = setOf(storage.uuid),
39

```

```

37         encryptionKind = StorageSyncGroupEncryptionKind.NONE,
38     )
39     assertFalse(isStorageCompatibleWithGroup(storage = storage, group = group))
40 }
41 }
42

```

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

```

1  package com.github.nullptroma.wallenc.usecases
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.StorageSyncPaths
6  import com.github.nullptroma.wallenc.domain.datatypes.StorageSyncRevision
7  import com.github.nullptroma.wallenc.domain.interfaces.StorageSyncGroup
8  import com.github.nullptroma.wallenc.domain.interfaces.StorageSyncGroupEncryptionKind
9  import com.github.nullptroma.wallenc.domain.tasks.TaskProgressLabel
10 import com.github.nullptroma.wallenc.usecases.fakes.FakeStorage
11 import com.github.nullptroma.wallenc.usecases.fakes.FakeStorageAccessor
12 import com.github.nullptroma.wallenc.usecases.fakes.FakeStorageSyncGroupStore
13 import com.github.nullptroma.wallenc.usecases.fakes.FakeVaultsManager
14 import kotlinx.coroutines.CancellationException
15 import kotlinx.coroutines.async
16 import kotlinx.coroutines.runBlocking
17 import org.junit.Assert.assertArrayEquals
18 import org.junit.Assert.assertEquals
19 import org.junit.Assert.assertNull
20 import org.junit.Assert.assertTrue
21 import org.junit.Test
22 import java.time.Instant
23
24 class StorageSyncEngineTest {
25
26     private fun norm(path: String): String = StorageSyncPaths.normalize(path)
27
28     @Test
29     fun syncAllGroupsReportsNoGroupsWhenEmpty() = runBlocking {
30         val labels = mutableListOf<TaskProgressLabel?>()
31         val engine = createEngine(storages = emptyList(), groups = emptyList())
32         engine.syncAllGroups { _, label -> labels.add(label) }
33         assertTrue(labels.any { it is TaskProgressLabel.SyncNoGroups })
34     }
35
36     @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),

```

```

82         encryptionKind = StorageSyncGroupEncryptionKind.NONE,
83     )
84     val labels = mutableList0f<TaskProgressLabel?>()
85     val engine = createEngine(storages = list0f(only), groups = list0f(group))
86     engine.syncGroup(groupId) { _, label -> labels.add(label) }
87     assertTrue(labels.any { it is TaskProgressLabel.SyncGroupSkippedTooFewStorages })
88 }
89
90 @Test
91 fun syncGroupDeleteRemovesFileOnTarget() = runBlocking {
92     val source = FakeStorage()
93     val target = FakeStorage()
94     val path = "obsolete.bin"
95     target.putFile(path, "old".encodeToByteArray())
96
97     val entry = StorageSyncJournalEntry(
98         path = path,
99         operation = StorageSyncOperation.DELETE,
100         revision = StorageSyncRevision(
101             sequence = 2L,
102             actorId = "actor-b",
103             createdAt = Instant.parse("2024-06-01T00:00:00Z"),
104         ),
105     )
106     source.addSyncJournalEntry(entry)
107
108     val group = StorageSyncGroup(
109         id = "delete-group",
110         storageUuids = set0f(source.uuid, target.uuid),
111         encryptionKind = StorageSyncGroupEncryptionKind.NONE,
112     )
113     val engine = createEngine(
114         storages = list0f(source, target),
115         groups = list0f(group),
116     )
117     engine.syncGroup(group.id) { _, _ -> }
118
119     assertNull(target.fileBytes(path))
120     val targetEntry = target.syncJournalEntries().single { it.path == norm(path) }
121     assertEquals(2L, targetEntry.revision.sequence)
122     assertEquals("actor-b", targetEntry.revision.actorId)
123     assertEquals(StorageSyncOperation.DELETE, targetEntry.operation)
124 }
125
126 @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

```

```

217     val group = StorageSyncGroup(
218         id = "trash-group",
219         storageUuids = setOf(source.uuid, target.uuid),
220         encryptionKind = StorageSyncGroupEncryptionKind.NONE,
221     )
222     val engine = createEngine(
223         storages = listOf(source, target),
224         groups = listOf(group),
225     )
226     engine.syncGroup(group.id) { _, _ -> }
227
228     assertArrayEquals(payload, target.fileBytes(path))
229     assertTrue(norm(path) in (target.accessor as FakeStorageAccessor).trashedPaths)
230     val targetEntry = target.syncJournalEntries().single { it.path == norm(path) }
231     assertEquals(3L, targetEntry.revision.sequence)
232     assertEquals("actor-trash", targetEntry.revision.actorId)
233     assertEquals(StorageSyncOperation.TRASH, targetEntry.operation)
234 }
235
236 @Test
237 fun syncGroupStopsWhenLockCannotBeAcquired() = runBlocking {
238     val first = FakeStorage()
239     val second = FakeStorage().apply { setAcquireLockResult(false) }
240     val group = StorageSyncGroup(
241         id = "lock-fail",
242         storageUuids = setOf(first.uuid, second.uuid),
243         encryptionKind = StorageSyncGroupEncryptionKind.NONE,
244     )
245     first.addSyncJournalEntry(
246         StorageSyncJournalEntry(
247             path = "a.txt",
248             operation = StorageSyncOperation.UPSERT,
249             revision = StorageSyncRevision(1L, "x", Instant.EPOCH),
250         ),
251     )
252     val labels = mutableList0f<TaskProgressLabel?>()
253     val engine = createEngine(
254         storages = listOf(first, second),
255         groups = listOf(group),
256     )
257     engine.syncGroup(group.id) { _, label -> labels.add(label) }
258     assertTrue(labels.any { it is TaskProgressLabel.SyncGroupLockFailed })
259     assertNull((first.accessor as FakeStorageAccessor).syncLock)
260     assertNull((second.accessor as FakeStorageAccessor).syncLock)
261 }

```

```

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()

```

```

352     val group = StorageSyncGroup(
353         id = "empty-journal",
354         storageUuids = setOf(first.uuid, second.uuid),
355         encryptionKind = StorageSyncGroupEncryptionKind.NONE,
356     )
357     val labels = mutableListOf<TaskProgressLabel?>()
358     val engine = createEngine(
359         storages = listOf(first, second),
360         groups = listOf(group),
361     )
362     engine.syncGroup(group.id) { _, label -> labels.add(label) }
363     assertTrue(labels.any { it is TaskProgressLabel.SyncGroupNoJournalEntries })
364     assertNull((first.accessor as FakeStorageAccessor).syncLock)
365     assertNull((second.accessor as FakeStorageAccessor).syncLock)
366 }
367
368 private fun createEngine(
369     storages: List<FakeStorage>,
370     groups: List<StorageSyncGroup>,
371 ): StorageSyncEngine {
372     val vaultsManager = FakeVaultsManager(storages)
373     val findStorage = FindStorageUseCase(vaultsManager)
374     return StorageSyncEngine(
375         vaultsManager = vaultsManager,
376         groupStore = FakeStorageSyncGroupStore(groups),
377         findStorageUseCase = findStorage,
378     )
379 }
380 }
381

```

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

```

1 package com.github.nullptroma.wallenc.usecases
2
3 import com.github.nullptroma.wallenc.domain.datatypes.StorageSyncJournalEntry
4 import com.github.nullptroma.wallenc.domain.datatypes.StorageSyncJournalMerge
5 import com.github.nullptroma.wallenc.domain.datatypes.StorageSyncOperation
6 import com.github.nullptroma.wallenc.domain.datatypes.StorageSyncPaths
7 import com.github.nullptroma.wallenc.domain.datatypes.StorageSyncRevision
8 import org.junit.Assert.assertEquals
9 import org.junit.Assert.assertFalse
10 import org.junit.Assert.assertTrue
11 import org.junit.Test
12 import java.time.Instant

```

```

13
14 class StorageSyncJournalMergeTest {
15
16     @Test
17     fun mergeKeepsSingleEntryPerPath() {
18         val older = entry(path = "/a.txt", sequence = 1L)
19         val newer = entry(path = "/a.txt", sequence = 2L)
20         val merged = StorageSyncJournalMerge.merge(
21             mapOf("/a.txt" to older),
22             mapOf("/a.txt" to newer),
23         )
24         assertEquals(1, merged.size)
25         assertEquals(2L, merged["/a.txt"]?.revision?.sequence)
26     }
27
28     @Test
29     fun isSyncableUserPathExcludesEncDirAndJournal() {
30         assertFalse(StorageSyncPaths.isSyncableUserPath("/88a048ed-enc-dir/sync-
journal.json"))
31         assertFalse(StorageSyncPaths.isSyncableUserPath("/wallenc-yandex-system/sync-
journal.json"))
32         assertTrue(StorageSyncPaths.isSyncableUserPath("/wallenc-data/text-secrets.json"))
33     }
34
35     private fun entry(path: String, sequence: Long): StorageSyncJournalEntry =
36         StorageSyncJournalEntry(
37             path = path,
38             operation = StorageSyncOperation.UPSERT,
39             revision = StorageSyncRevision(
40                 sequence = sequence,
41                 actorId = "actor",
42                 createdAt = Instant.EPOCH,
43             ),
44         )
45 }
46

```

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

```

1 package com.github.nullptroma.wallenc.usecases
2
3 import com.github.nullptroma.wallenc.domain.datatypes.TwoFaTokenRecord
4 import com.eatthepath.otp.TimeBasedOneTimePasswordGenerator
5 import org.junit.Assert.assertEquals
6 import org.junit.Assert.assertNotNull
7 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),
43                 0.001f)
44         }
45
46         @Test
47         fun totpSecondsUntilRefreshCountsDownWithinPeriod() {
48             val period = 30
49             val t0 = 1_700_000_100_000L
50             assertEquals(30, totpSecondsUntilRefresh(t0, period))
51             assertEquals(15, totpSecondsUntilRefresh(t0 + 15_000L, period))
52             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

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

```
1 package com.github.nullptroma.wallenc.ui.screens.main.screens.storage.secrets
2
3 import androidx.compose.material3.MaterialTheme
4 import androidx.compose.ui.test.assertIsDisplayed
5 import androidx.compose.ui.test.junit4.createComposeRule
6 import androidx.compose.ui.test.onNodeWithText
7 import androidx.test.ext.junit.runners.AndroidJUnit4
8 import androidx.test.platform.app.InstrumentationRegistry
9 import com.github.nullptroma.wallenc.domain.datatypes.TextSecretEntryRecord
10 import com.github.nullptroma.wallenc.domain.datatypes.TextSecretRecord
11 import com.github.nullptroma.wallenc.ui.R
12 import org.junit.Rule
13 import org.junit.Test
14 import org.junit.runner.RunWith
15
16 @RunWith(AndroidJUnit4::class)
17 class TextSecretsScreenContentTest {
18
19     @get:Rule
20     val composeRule = createComposeRule()
21
22     private val context get() = InstrumentationRegistry.getInstrumentation().targetContext
23
24     @Test
25     fun showsEmptyStateWhenNoSecrets() {
26         composeRule.setContent {
27             MaterialTheme {
28                 TextSecretsScreenContent(
29                     uiState = TextSecretsScreenState(
30                         isLoading = false,
31                         isAvailable = true,
32                         items = emptyList(),
33                     ),
34                 )
35             }
36         }
37         composeRule.onNodeWithText(context.getString(R.string.text_secret_empty_state)).assertIsDisplayed()
38     }
39
40     @Test
41     fun showsSecretTitle() {
42         val secret = TextSecretRecord(
```

```

43         id = "s1",
44         title = "Production DB",
45         items = listOf(TextSecretEntryRecord(label = "user", value = "admin")),
46     )
47     composeRule.setContent {
48         MaterialTheme {
49             TextSecretsScreenContent(
50                 uiState = TextSecretsScreenState(
51                     isLoading = false,
52                     isAvailable = true,
53                     items = listOf(secret),
54                 ),
55             ) {
56                 TextSecretListCard(
57                     secret = secret,
58                     onClick = {},
59                     enabled = true,
60                 )
61             }
62         }
63     }
64     composeRule.onNodeWithText("Production DB").assertIsDisplayed()
65 }
66 }
67

```

ABTOTECT ui/src/androidTest/java/com/github/nullptroma/wallenc/ui/screens/main/  
screens/storage/twoFa/TwoFaTokensScreenContentTest.kt

```

1     package com.github.nullptroma.wallenc.ui.screens.main.screens.storage.twoFa
2
3     import androidx.compose.foundation.layout.padding
4     import androidx.compose.material3.MaterialTheme
5     import androidx.compose.ui.Modifier
6     import androidx.compose.ui.test.junit4.createComposeRule
7     import androidx.compose.ui.test.onNodeWithText
8     import androidx.compose.ui.test.assertIsDisplayed
9     import androidx.compose.ui.unit.dp
10    import androidx.test.ext.junit.runners.AndroidJUnit4
11    import androidx.test.platform.app.InstrumentationRegistry
12    import com.github.nullptroma.wallenc.domain.datatypes.TwoFaTokenRecord
13    import com.github.nullptroma.wallenc.ui.R
14    import org.junit.Rule
15    import org.junit.Test
16    import org.junit.runner.RunWith
17

```

```

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),

```

```

63         )
64     }
65 }
66 }
67 composeRule.onNodeWithText("GitHub").assertIsDisplayed()
68 composeRule.onNodeWithText("user@example.com").assertIsDisplayed()
69 }
70 }
71

```

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

```

1  package com.github.nullptroma.wallenc.ui.navigation
2
3  import android.content.Intent
4  import android.net.Uri
5  import org.junit.Assert.assertFalse
6  import org.junit.Assert.assertTrue
7  import org.junit.Test
8  import org.junit.runner.RunWith
9  import org.robolectric.RobolectricTestRunner
10 import org.robolectric.annotation.Config
11
12 @RunWith(RobolectricTestRunner::class)
13 @Config(sdk = [33])
14 class WallencDeepLinksTest {
15
16     @Test
17     fun matchesWallencViewIntent() {
18         val intent = Intent(Intent.ACTION_VIEW,
19 Uri.parse(WallencDeepLinks.MAIN_URI_PATTERN))
20         assertTrue(intent.matchesWallencDeepLink())
21     }
22
23     @Test
24     fun rejectsUnrelatedIntent() {
25         val intent = Intent(Intent.ACTION_MAIN)
26         assertFalse(intent.matchesWallencDeepLink())
27     }
28
29     @Test
30     fun matchesTasksAndSettingsHosts() {
31         assertTrue(
32             Intent(Intent.ACTION_VIEW, Uri.parse(WallencDeepLinks.TASKS_URI_PATTERN))
33                 .matchesWallencDeepLink(),
34         )
35     }
36 }
37

```

```

34         assertTrue(
35             Intent(Intent.ACTION_VIEW, Uri.parse(WallencDeepLinks.SETTINGS_URI_PATTERN))
36                 .matchesWallencDeepLink(),
37         )
38     }
39 }
40

```

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

```

1  package com.github.nullptroma.wallenc.ui.resources
2
3  import com.github.nullptroma.wallenc.domain.tasks.TaskProgressLabel
4  import com.github.nullptroma.wallenc.domain.tasks.VaultTaskStep
5  import com.github.nullptroma.wallenc.ui.R
6  import org.junit.Assert.assertEquals
7  import org.junit.Test
8
9  class TaskProgressLabelsTest {
10
11      private val resolver = object : UiStringResolver {
12          override fun invoke(id: Int, vararg formatArgs: Any): String = "res:$id"
13
14          override fun plurals(id: Int, quantity: Int, vararg formatArgs: Any): String =
15              "plural:$id"
16      }
17
18      @Test
19      fun syncNoGroups_mapsToStringRes() {
20          val text = TaskProgressLabel.SyncNoGroups.resolve(resolver)
21          assertEquals("res:${R.string.sync_progress_no_groups}", text)
22      }
23
24      @Test
25      fun vaultTask_mapsToStringRes() {
26          val text = TaskProgressLabel.VaultTask(VaultTaskStep.Save2FaToken).resolve(resolver)
27          assertEquals("res:${R.string.task_progress_save_2fa_token}", text)
28      }
29
30      @Test
31      fun clearContentProgress_mapsToStringRes() {
32          val text = TaskProgressLabel.ClearContentProgress(3, 10).resolve(resolver)
33          assertEquals("res:${R.string.task_progress_clear_content}", text)
34      }
35
36  }

```

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

```

1  package com.github.nullptroma.wallenc.ui.resources
2
3  import com.github.nullptroma.wallenc.domain.errors.WallencException
4  import com.github.nullptroma.wallenc.ui.R
5  import org.junit.Assert.assertEquals
6  import org.junit.Test
7
8  class WallencUserNotificationMappingTest {
9
10     @Test
11     fun mapsFeatureStorageNotFound() {
12         val notification = WallencException.Feature.StorageNotFound().toUserNotification()
13         assertEquals(R.string.error_storage_not_found, notification.id)
14     }
15
16     @Test
17     fun mapsStorageIncorrectKey() {
18         val notification = WallencException.Storage.IncorrectKey().toUserNotification()
19         assertEquals(R.string.error_incorrect_password, notification.id)
20     }
21
22     @Test
23     fun mapsUnknown() {
24         val notification = WallencException.Unknown(Exception("x")).toUserNotification()
25         assertEquals(R.string.error_unknown, notification.id)
26     }
27 }
28

```

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

```

1  package com.github.nullptroma.wallenc.ui.screens.main.screens.storage
2
3  import
4  com.github.nullptroma.wallenc.ui.screens.main.screens.storage.secrets.TextSecretDetailsRoute
5  import
6  com.github.nullptroma.wallenc.ui.screens.main.screens.storage.secrets.TextSecretEditRoute
7  import
8  com.github.nullptroma.wallenc.ui.screens.main.screens.storage.secrets.TextSecretsRoute
9  import com.github.nullptroma.wallenc.ui.screens.main.screens.storage.twofa.TwoFaTokensRoute
10 import org.junit.Assert.assertEquals
11 import org.junit.Assert.assertNull

```



```

9      import org.junit.Test
10
11      class StorageNavigationRoutesSmokeTest {
12
13          @Test
14          fun storageHomeRouteCarriesVaultAndStorageIds() {
15              val route = StorageHomeRoute(
16                  vaultUuid = "vault-1",
17                  storageUuid = "storage-1",
18              )
19              assertEquals("vault-1", route.vaultUuid)
20              assertEquals("storage-1", route.storageUuid)
21          }
22
23          @Test
24          fun textSecretsRoutesCarryRequiredArguments() {
25              val listRoute = TextSecretsRoute(storageUuid = "storage-1")
26              val detailsRoute = TextSecretDetailsRoute(storageUuid = "storage-1", secretId =
"secret-1")
27              val editRoute = TextSecretEditRoute(storageUuid = "storage-1", secretId = null)
28              val twoFaRoute = TwoFaTokensRoute(storageUuid = "storage-1")
29
30              assertEquals("storage-1", listRoute.storageUuid)
31              assertEquals("secret-1", detailsRoute.secretId)
32              assertNull(editRoute.secretId)
33              assertEquals("storage-1", twoFaRoute.storageUuid)
34          }
35      }
36

```

ABTOTECT ui/src/test/java/com/github/nullptroma/wallenc/ui/screens/main/screens/  
storage/twoFa/OtpAuthUriParserTest.kt

```

1      package com.github.nullptroma.wallenc.ui.screens.main.screens.storage.twoFa
2
3      import org.junit.Assert.assertEquals
4      import org.junit.Assert.assertNotNull
5      import org.junit.Assert.assertNull
6      import org.junit.Test
7      import org.junit.runner.RunWith
8      import org.robolectric.RobolectricTestRunner
9      import org.robolectric.annotation.Config
10
11      @RunWith(RobolectricTestRunner::class)
12      @Config(sdk = [33])
13      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

```

ABTOTECT 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

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

```
1 package com.github.nullptroma.wallenc.task.runtime
2
3 import com.github.nullptroma.wallenc.domain.errors.WallencException
4 import com.github.nullptroma.wallenc.domain.tasks.TaskLogLevel
5 import com.github.nullptroma.wallenc.domain.tasks.TaskProgressLabel
6 import com.github.nullptroma.wallenc.domain.tasks.TaskRunState
7 import kotlinx.coroutines.ExperimentalCoroutinesApi
8 import kotlinx.coroutines.test.StandardTestDispatcher
9 import kotlinx.coroutines.test.advanceTimeBy
10 import kotlinx.coroutines.test.advanceUntilIdle
11 import kotlinx.coroutines.test.runTest
12 import org.junit.Assert.assertTrue
13 import org.junit.Test
14
15 @OptIn(ExperimentalCoroutinesApi::class)
16 class TaskOrchestratorTest {
17
18     private val dispatcher = StandardTestDispatcher()
19
20     @Test
21     fun enqueueCompletesTask() = runTest(dispatcher) {
22         val orchestrator = TaskOrchestrator(dispatcher)
23         val id = orchestrator.enqueue(
24             title = "Test",
25             dispatcher = dispatcher,
26             work = { ctx ->
27                 ctx.reportProgress(0.5f, TaskProgressLabel.SyncPreparing(1))
28             },
29         )
30         advanceUntilIdle()
31         val task = orchestrator.pipelineState.value.tasks.first { it.id == id }
32         assertTrue(task.state is TaskRunState.Completed)
33     }
34
35     @Test
36     fun cancelAllMarksRunningTaskCancelled() = runTest(dispatcher) {
37         val orchestrator = TaskOrchestrator(dispatcher)
38         val id = orchestrator.enqueue(
39             title = "Long",
40             dispatcher = dispatcher,
41             work = { ctx ->
42                 ctx.reportProgress(null, null)
43                 kotlinx.coroutines.delay(60_000)
44             }
45         )
46     }
```

```

44         },
45     )
46     advanceTimeBy(1)
47     orchestrator.cancelAll()
48     advanceUntilIdle()
49     val task = orchestrator.pipelineState.value.tasks.first { it.id == id }
50     assertTrue(task.state is TaskRunState.Cancelled)
51 }
52
53 @Test
54 fun cancelMarksTaskCancelled() = runTest(dispatcher) {
55     val orchestrator = TaskOrchestrator(dispatcher)
56     val id = orchestrator.enqueue(
57         title = "Long",
58         dispatcher = dispatcher,
59         work = { ctx ->
60             ctx.reportProgress(null, null)
61             kotlinx.coroutines.delay(60_000)
62         },
63     )
64     advanceTimeBy(1)
65     orchestrator.cancel(id)
66     advanceUntilIdle()
67     val task = orchestrator.pipelineState.value.tasks.first { it.id == id }
68     assertTrue(task.state is TaskRunState.Cancelled)
69 }
70
71 @Test
72 fun failRecordsFailedState() = runTest(dispatcher) {
73     val orchestrator = TaskOrchestrator(dispatcher)
74     val id = orchestrator.enqueue(
75         title = "Fail",
76         dispatcher = dispatcher,
77         work = { ctx ->
78             ctx.fail(WallencException.Storage.FileNotFound())
79         },
80     )
81     advanceUntilIdle()
82     val task = orchestrator.pipelineState.value.tasks.first { it.id == id }
83     val failed = task.state as TaskRunState.Failed
84     assertTrue(failed.error is WallencException.Storage.FileNotFound)
85 }
86
87 @Test
88 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

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

```
1     package com.github.nullptroma.wallenc.infrastructure.android.db.app.repository
2
3     import androidx.room.Room
4     import androidx.test.core.app.ApplicationProvider
5     import androidx.test.ext.junit.runners.AndroidJUnit4
6     import com.github.nullptroma.wallenc.domain.vault.model.YandexAccount
7     import com.github.nullptroma.wallenc.infrastructure.android.db.app.AppDb
8     import kotlinx.coroutines.Dispatchers
9     import kotlinx.coroutines.runBlocking
10    import org.junit.After
11    import org.junit.Assert.assertEquals
12    import org.junit.Assert.assertNull
13    import org.junit.Before
14    import org.junit.Test
15    import org.junit.runner.RunWith
16
17    @RunWith(AndroidJUnit4::class)
18    class YandexAccountRepositoryTest {
19
20        private lateinit var db: AppDb
21        private lateinit var repository: YandexAccountRepository
22
23        @Before
24        fun setUp() {
25            val context = ApplicationProvider.getApplicationContext<android.content.Context>()
26            db = Room.inMemoryDatabaseBuilder(context, AppDb::class.java).build()
27            repository = YandexAccountRepository(
28                dao = db.yandexAccountDao,
29                ioDispatcher = Dispatchers.IO,
30            )
31        }
32
33        @After
34        fun tearDown() {
35            db.close()
36        }
37
38        @Test
39        fun insertAndLoadByVaultUuid() = runBlocking {
40            val account = YandexAccount(
41                vaultUuid = "vault-1",
42                yandexUserId = "user-1",
43                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

```
1 package com.github.nullptroma.wallenc.app.integration.yandex
2
3 import com.github.nullptroma.wallenc.domain.vault.network.yandexdisk.YandexDiskApiFactory
4 import com.github.nullptroma.wallenc.domain.vault.network.yandexdisk.repository.YandexDiskRepository
5 import kotlinx.coroutines.Dispatchers
6 import kotlinx.coroutines.runBlocking
7 import org.junit.After
8 import org.junit.Assert.assertNotNull
9 import org.junit.Assert.assertTrue
10 import org.junit.Before
11 import org.junit.Test
12 import org.junit.runner.RunWith
13 import org.junit.runners.JUnit4
14
15 /**
16  * Live-прогон Disk API. Пути только `app:/` — как в [YandexVault]; токен из Auth SDK
17  * обычно имеет `cloud_api:disk.app_folder`, без полного доступа к `disk:/` (там 403).
18  */
19 @RunWith(JUnit4::class)
20 class YandexDiskLiveIntegrationTest {
21
22     private lateinit var repository: YandexDiskRepository
23     private val testFolder = "app:/wallenc-integration-test"
24     private val probeFileName = "wallenc-probe.txt"
25     private val probePath = "$testFolder/$probeFileName"
26     private val probePayload = "wallenc-integration-probe".encodeToByteArray()
27
28     @Before
29     fun setUp() {
30         YandexTestCredentials.assumePresent()
31         val token = YandexTestCredentials.oauthToken()!!
32         repository = YandexDiskApiFactory.createRepositoryWithToken(token, Dispatchers.IO)
33         runBlocking {
34             runCatching { repository.createFolder(testFolder) }
35             runCatching {
36                 repository.uploadBytes(probePath, probePayload, overwrite = true)
37             }
38         }
39     }
40
41     @After
42     fun tearDown(): Unit = runBlocking {
```

```

43         runCatching { repository.delete(probePath, permanently = true) }
44     }
45
46     @Test
47     fun diskInfoReturnsQuota() = runBlocking {
48         val info = repository.diskInfo()
49         assertNotNull(info.totalSpace)
50         assertTrue(info.totalSpace!! > 0)
51     }
52
53     @Test
54     fun listTestFolderDoesNotThrow() = runBlocking {
55         val result = repository.list(testFolder, limit = 10, offset = 0)
56         assertNotNull(result)
57     }
58
59     @Test
60     fun uploadAndDownloadRoundTrip() = runBlocking {
61         val path = "$testFolder/roundtrip-${System.currentTimeMillis()}.bin"
62         try {
63             repository.uploadBytes(path, probePayload, overwrite = true)
64             val downloaded = repository.openDownloadStream(path).use { it.readBytes() }
65             assertTrue(downloaded.contentEquals(probePayload))
66         } finally {
67             runCatching { repository.delete(path, permanently = true) }
68         }
69     }
70 }
71

```

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

```

1     package com.github.nullptroma.wallenc.app.integration.yandex
2
3     import androidx.test.platform.app.InstrumentationRegistry
4     import org.junit.Assume.assumeFalse
5
6     object YandexTestCredentials {
7         fun oauthToken(): String? =
8             InstrumentationRegistry.getArguments().getString("yandex.oauth.token")?.takeIf
9             { it.isNotBlank() }
10
11         fun assumePresent(message: String = "Добавьте yandex.test.oauth.token в
12         local.properties") {
13             assumeFalse(message, oauthToken().isNullOrBlank())
14         }
15     }
16

```

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.

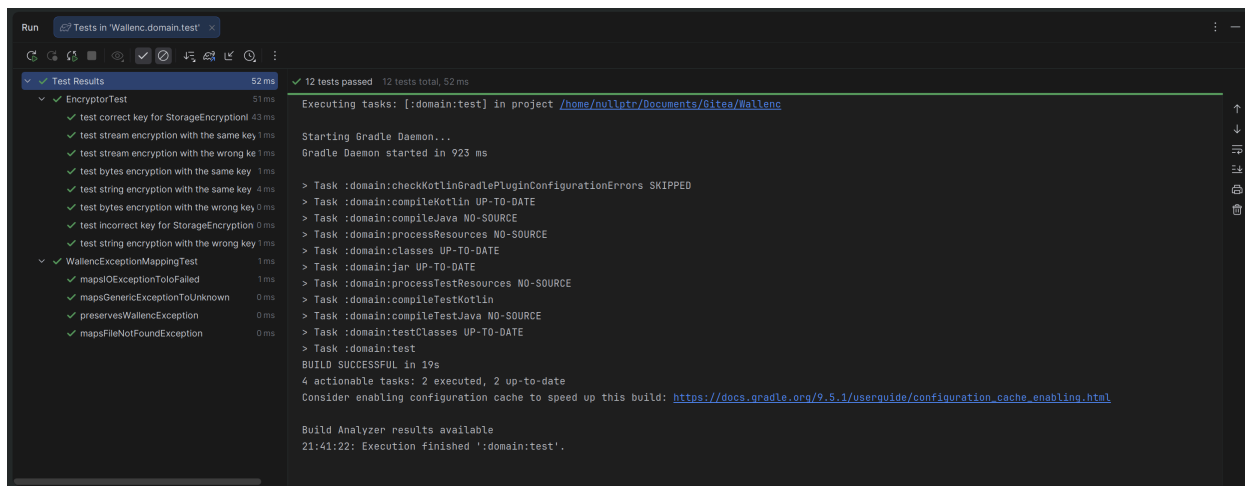


Рис. 1. Отчёт Gradle: модуль `:domain`, задача `test`

### 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.

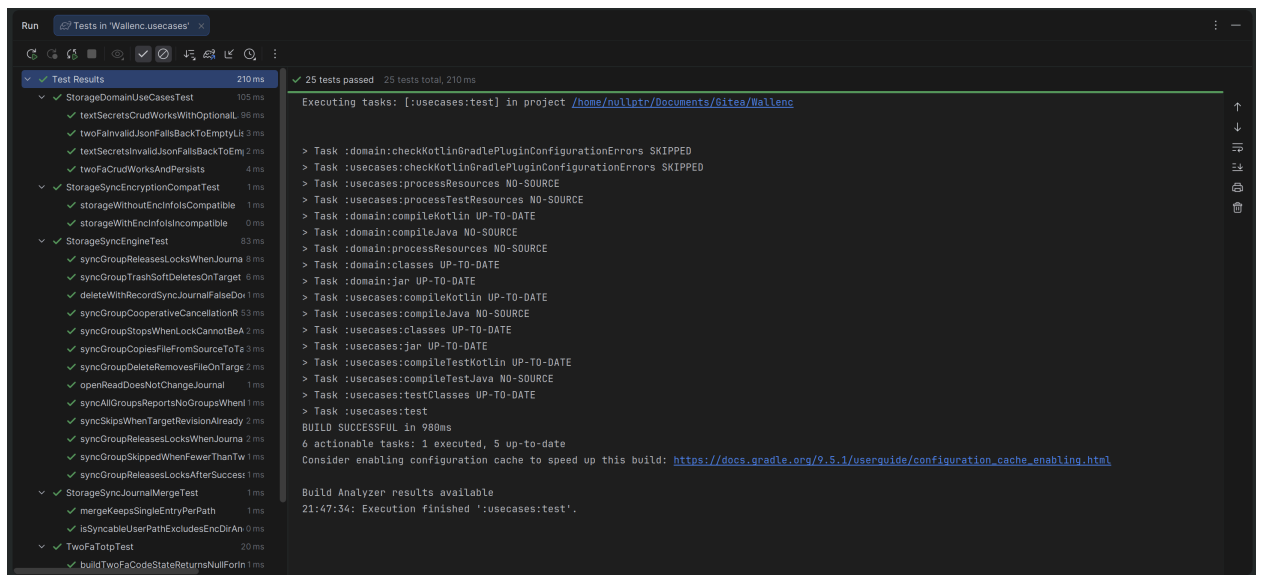


Рис. 2. Отчёт Gradle: модуль :usecases

### 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).

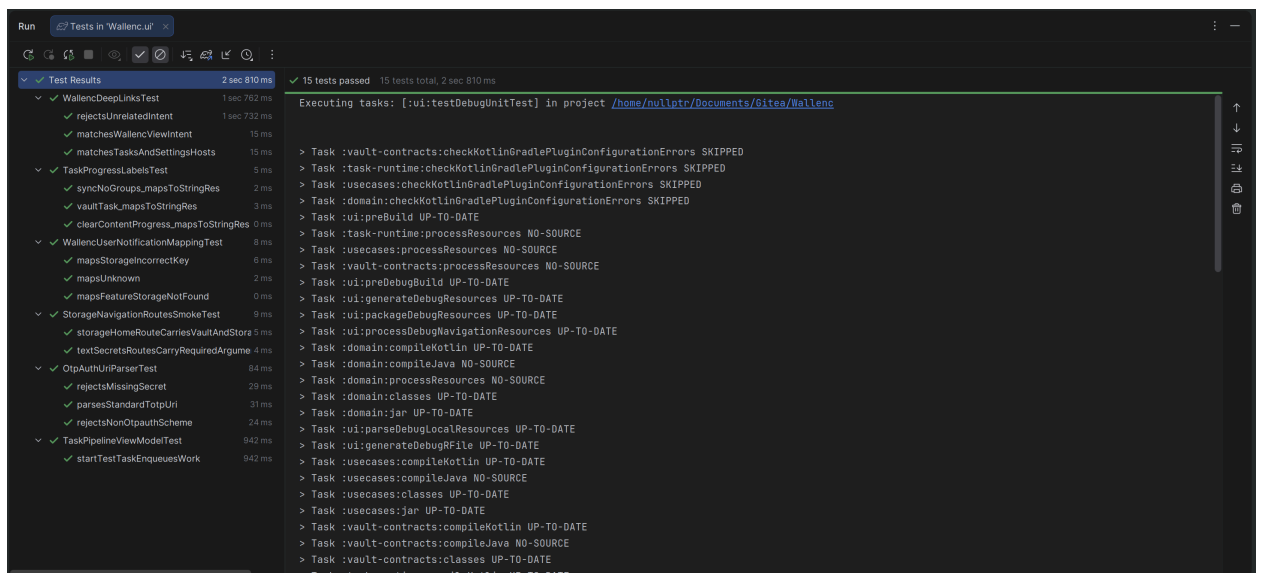


Рис. 3. Отчёт Gradle: модуль :ui

### 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.

```
✖ nullptr@thinkbook ~/Documents/Gitea/Wallenc ▶ r main ⇨ ▶ ./gradlew connectedDebugAndroidTest
Starting 3 tests on Medium_Phone_API_36.1(AVD) - 16
Finished 3 tests on Medium_Phone_API_36.1(AVD) - 16
Starting 4 tests on Medium_Phone_API_36.1(AVD) - 16
Finished 4 tests on Medium_Phone_API_36.1(AVD) - 16
Starting 3 tests on Medium_Phone_API_36.1(AVD) - 16
Medium_Phone_API_36.1(AVD) - 16 Tests 1/3 completed. (0 skipped) (0 failed)
Medium_Phone_API_36.1(AVD) - 16 Tests 2/3 completed. (0 skipped) (0 failed)
Finished 3 tests on Medium_Phone_API_36.1(AVD) - 16

BUILD SUCCESSFUL in 55s
193 actionable tasks: 19 executed, 174 up-to-date
Consider enabling configuration cache to speed up this build: https://docs.gradle.org/9.5.1/userguide/configuration\_cache\_enabling.html
nullptr@thinkbook ~/Documents/Gitea/Wallenc ▶ r main ⇨ ▶
```

Рис. 4. Gradle connectedDebugAndroidTest