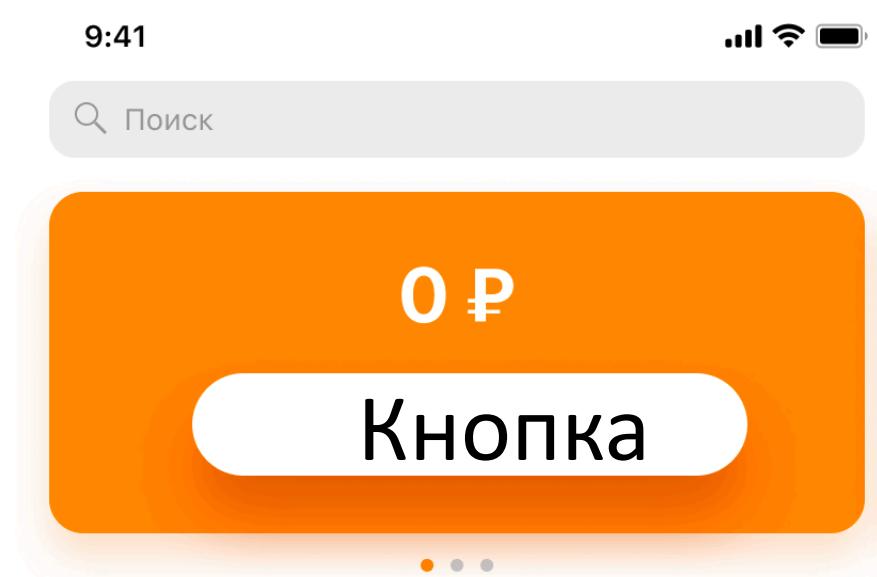




Сага о распределённых транзакциях

Котенев Андрей

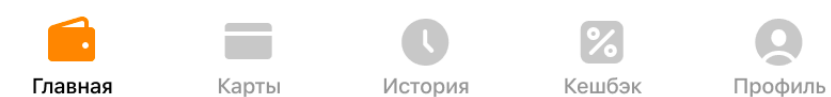
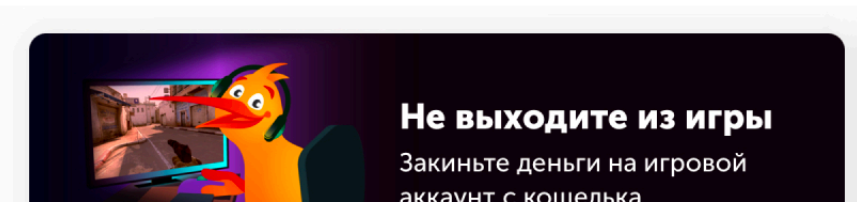
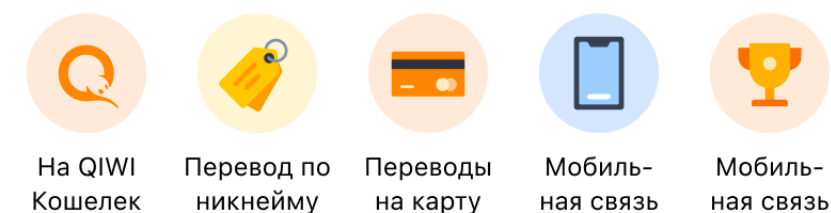
Получение услуги

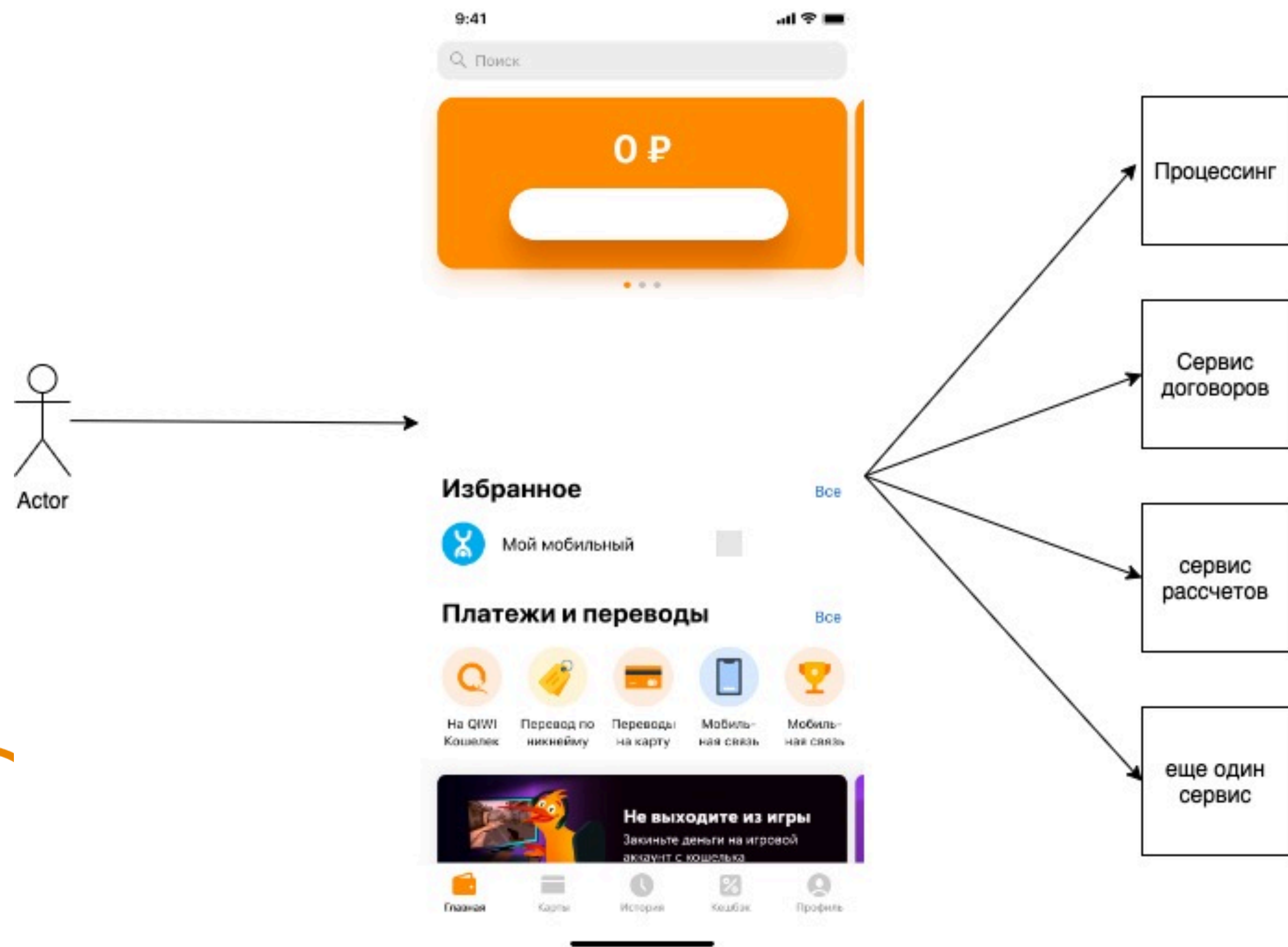


Избранное [Все](#)



Платежи и переводы [Все](#)





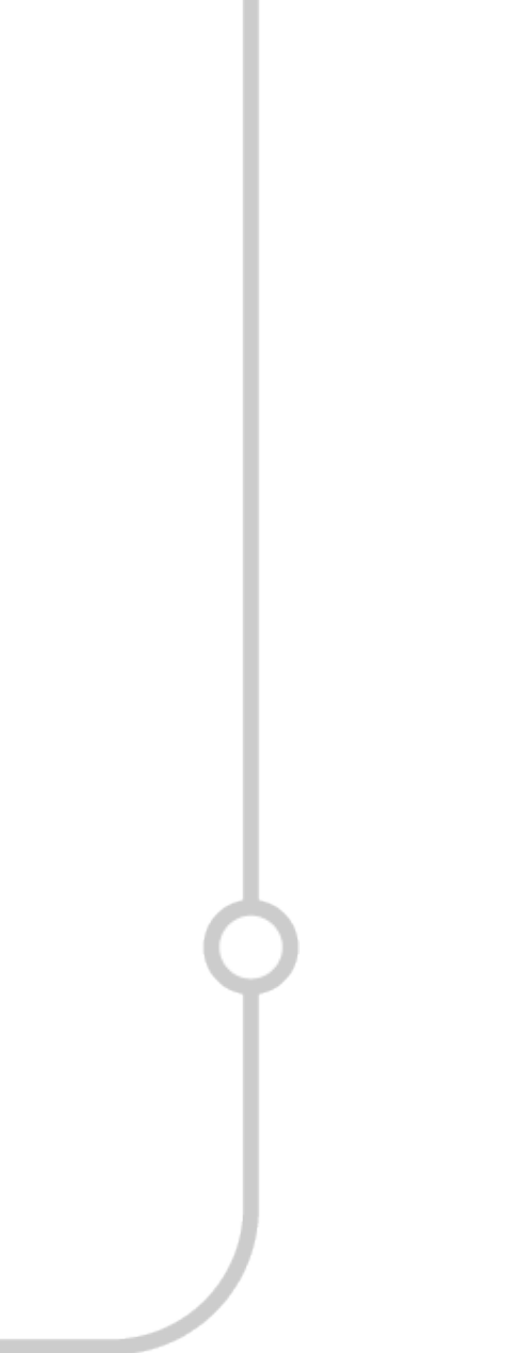
Saga

Бизнес процесс разбит на локальные шаги, для каждого из которых определено компенсирующее действие. Список шагов фиксируется, в случае ошибки - выполняем компенсирующие операции

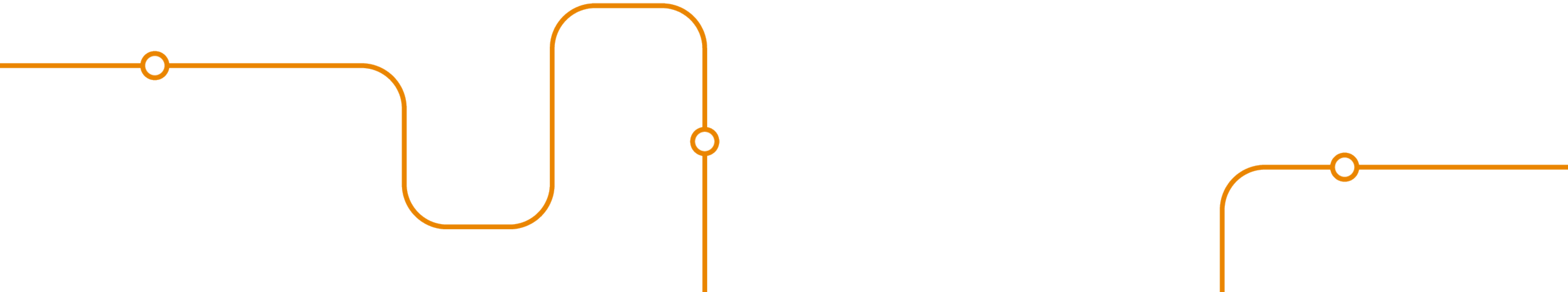
<https://microservices.io/patterns/data/saga.html>

A decorative orange line starts from the left edge, moves horizontally to the right, then curves down, then up, then down again, ending at the bottom edge. A small white circle with an orange outline is located on the first horizontal segment.


Операция должна быть доведена до конечного состояния или компенсировать выполненные шаги.




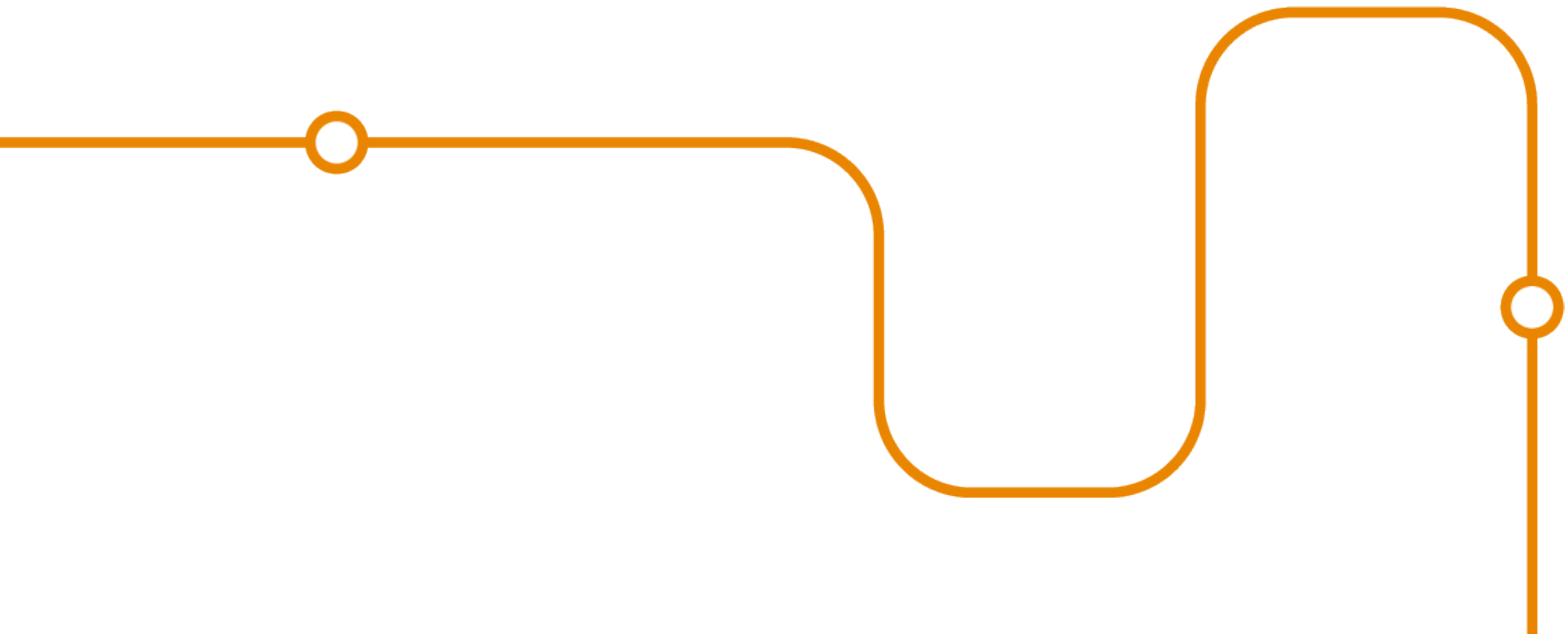
```
private fun process(flowRequest: FlowRequest): OperationResponse =  
    kotlin.runCatching {  
        lockHelperService.lock(flowRequest.client).use { it: AutoCloseableUnchecked  
            flowRequestModifiers.forEach { it.modify(flowRequest) }  
  
            getChain(flowRequest).process(flowRequest).toOperationResponse() ^use  
        }  
    }.getOrElse { it: Throwable  
        throw operationErrorProcessor.process(flowRequest, it)  
    }  
}
```



```
error-processing:
  error-recovery-policies:
    request_parameter_undefined: manual
    request_parameter_invalid: manual
    lock_failed: "periodical { period-seconds : 30 }"
    lock_timeout: "periodical { period-seconds : 30 }"
    otp_transport_failed: manual
    otp_incorrect: manual
    otp_resend_unavailable: manual
    sms_sending_failed: "progressive { min-period-seconds: 30, max-period-seconds : 86400, factor: 2 }"
    client_credit_limit_invalid: "periodical { period-seconds : 86400, max-attempts : 31 }"
    client_balance_not_found: "periodical { period-seconds : 86400, max-attempts : 31 }"
    client_balance_not_enough: "periodical { period-seconds : 86400, max-attempts : 31 }"
    client_card_invalid: "periodical { period-seconds : 86400, max-attempts : 31 }"
    banking_gateway_not_available: "progressive { min-period-seconds: 30, max-period-seconds : 86400, factor: 2 }"
```



Отдельные шаги операции реализуются в виде chain
Состояние выполнения операции сохраняется в бд
Имеется система восстановления с политиками *recovery*
Если мы не можем добыть операцию – можем запустить отдельную операцию восстановления





Система основана на конечном автомате

Логика выполняется во время смены состояния, события меняют состояние системы

Логика восстановления должна быть реализована опираясь на предыдущий опыт

Автомат может долгое время оставаться в некотором промежуточном состоянии

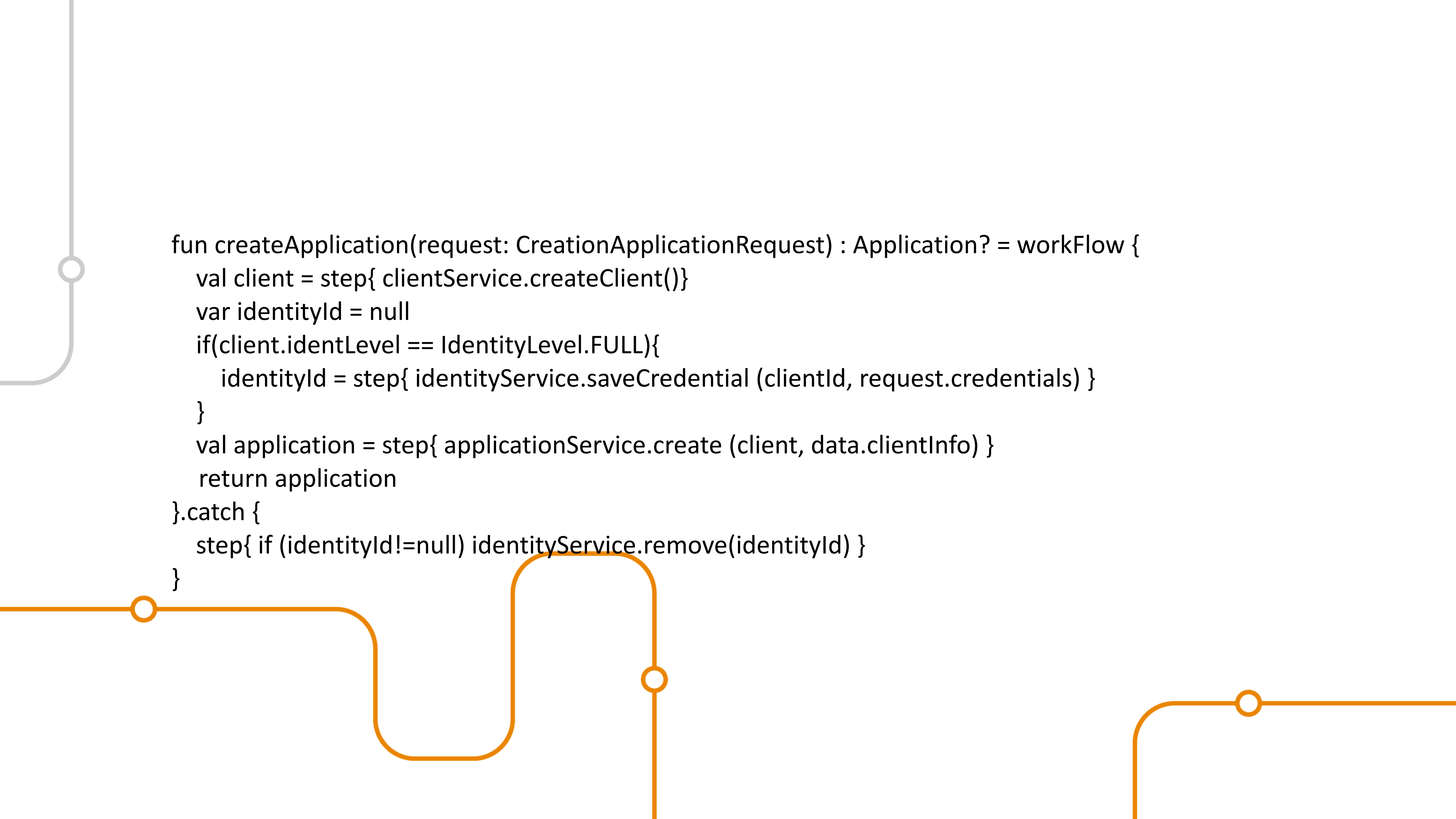
Должен быть DSL для описания состояний переходов и логики восстановления



```

init {
  sagaDefinition { this: SagaDslBuilder
    step { this: SagaDslStepBuilder
      state = TestSagaState.INITIAL_STATE
      transitions { this: TransitionsDsl
        transition { this: TransitionsBuilder
          event = TestSagaEventType.START_EVENT
          transitionAction = TransitionAction { _, _ -> TestSagaState.PROCESS_STATE }
          action = startTestAction
        }
        transition { this: TransitionsBuilder
          event = TestSagaEventType.FAILED_EVENT
          transitionAction = TransitionAction { _, _ -> TestSagaState.FAILED_STATE }
          action = rollbackTestAction
        }
      }
      catch { this: CatchDsl
        withErrorCodes { this: ErrorCodesCatchCaseBuilder
          errorCodes = listOf(TestsError.RECOVERABLE_IMMEDIATELY.code)
          recoveryPolicy = RecoveryPolicy.None
          recoveryAction = recoveryTestAction
        }
        withErrorCodes { this: ErrorCodesCatchCaseBuilder
          errorCodes = listOf(TestsError.RECOVERABLE_PERIODICAL.code)
          recoveryPolicy = RecoveryPolicy.Periodical( periodSeconds: 4, maxAttempts: 5)
          recoveryAction = recoveryTestAction
        }
        withErrorCodes { this: ErrorCodesCatchCaseBuilder
          errorCodes = listOf(TestsError.RECOVERABLE_PROGRESSIVE.code)
          recoveryPolicy = RecoveryPolicy.Progressive( minPeriodSeconds: 1, maxPeriodSeconds: 50, factor: 2.0, maxAttempts: 3)
          recoveryAction = recoveryTestAction
        }
        withCondition { this: ErrorConditionCatchCaseBuilder
          errorCondition = { it.isRetrievable() }
          recoveryPolicy = RecoveryPolicy.None
          recoveryAction = recoveryTestAction
        }
      }
    }
  }
  step { this: SagaDslStepBuilder

```

A decorative orange line with circles starts from the left edge, goes down, then right, then up, then right again, ending on the right edge. There are three circles at the corners of these segments.

```
fun createApplication(request: CreationApplicationRequest) : Application? = workFlow {  
    val client = step{ clientService.createClient()}  
    var identityId = null  
    if(client.identityLevel == IdentityLevel.FULL){  
        identityId = step{ identityService.saveCredential (clientId, request.credentials) }  
    }  
    val application = step{ applicationService.create (client, data.clientInfo) }  
    return application  
}.catch {  
    step{ if (identityId!=null) identityService.remove(identityId) }  
}
```

Спасибо за внимание !

