Алгоритмы работы с банковскими картами

Обзор системы

Система управления банковскими картами реализует комплексные алгоритмы для безопасного создания, управления и выполнения операций с картами. Все операции используют многоуровневое шифрование и строгую систему контроля доступа на основе ролей пользователей.

Создание новой карты

Алгоритм создания

1. **Проверка прав доступа**: Только администраторы могут создавать карты (`@PreAuthorize("hasRole('ROLE_ADMIN')")`)
2. **Валидация пользователя**: Проверка существования владельца карты в базе данных
3. **Генерация уникального номера**:
   - Генерация 16-значного номера через `ThreadLocalRandom`
   - Проверка уникальности через HMAC-хеш (до 100 попыток)
   - Сохранение хеша в таблице `CardHash`
4. **Двойное шифрование номера**:
   - Генерация индивидуального AES-ключа
   - Шифрование номера карты этим ключом
   - Шифрование самого ключа мастер-ключом
5. **Установка срока действия**: Последний день месяца через указанное количество месяцев (по умолчанию 24)
6. **Сохранение в базе**: Транзакционное сохранение карты и зашифрованного ключа

Структура номера карты

- **Формат**: 16 цифр в формате XXXX-XXXX-XXXX-XXXX
- **Генерация**: 4 блока по 4 случайные цифры
- **Уникальность**: Гарантируется через проверку HMAC-хеша

Поиск и идентификация карт

Поиск по номеру (`getCardByNumber`)

1. **Получение пользователя**: Извлечение ID из `SecurityContext`
2. **Загрузка карт пользователя**: Получение всех карт владельца из БД
3. **Итерация и расшифровка**:
   - Для каждой карты расшифровка её ключа мастер-ключом
   - Шифрование входящего номера найденным ключом
   - Сравнение зашифрованных значений
4. **Возврат результата**: Найденная карта или исключение `NotFoundException`

Оптимизация поиска

- **Ограничение области**: Поиск только среди карт текущего пользователя
- **Ленивая загрузка**: Расшифровка только при необходимости сравнения
- **Обработка ошибок**: Маскирование деталей через общие `RuntimeException`

Операции с балансом

Пополнение счета (`depositMoney`)

1. **Валидация суммы**: Проверка на отрицательные значения
2. **Проверка доступности карты**: Статус `ACTIVE` и действующий срок
3. **Проверка переполнения**: `currentBalance > Double.MAX_VALUE - sum`
4. **Обновление баланса**: Атомарная операция в базе данных
5. **Возврат результата**: Новый баланс в `CardBalanceResponse`

Снятие средств (`withdrawMoney`)

1. **Валидация суммы**: Проверка на отрицательные значения
2. **Проверка доступности карты**: Активная карта с действующим сроком
3. **Проверка достаточности средств**: `newBalance < 0`
4. **Обновление баланса**: Транзакционное уменьшение суммы
5. **Возврат результата**: Обновленный баланс

Денежные переводы (`doMoneyTransfer`)

1. **Валидация суммы перевода**: Отсутствие отрицательных значений
2. **Поиск карт**: Получение карт отправителя и получателя
3. **Проверка доступности**: Обе карты должны быть активными
4. **Двойная валидация баланса**:
   - Проверка возможности списания с карты отправителя
   - Проверка возможности зачисления на карту получателя

5. **Атомарная операция**: `cardRepository.transferMoney()` в одной транзакции

Система блокировки карт

Подача заявки на блокировку (`submitCardBlocking`)

1. **Поиск карты**: Получение карты по номеру
2. **Проверка доступности**: Карта должна быть активной для подачи заявки
3. **Создание заявки**: Сохранение `CardBlockingRequest` в базе
4. **Уведомление**: Заявка поступает в очередь для администраторов

Обработка заявок администратором (`processBlockRequest`)

1. **Проверка прав**: Только администраторы (`@PreAuthorize("hasRole('ROLE_ADMIN')")`)
2. **Валидация ID карты**: Проверка на `null` и существование в БД
3. **Блокировка карты**: Изменение статуса на `Card.Status.BLOCKED`
4. **Удаление заявки**: Очистка `CardBlockingRequest` после обработки

Получение списка заявок (`getBlockRequests`)

- **Пагинация**: Поддержка постраничного вывода
- **Права доступа**: Только для администраторов
- **Формат вывода**: Список ID карт с активными заявками

Управление жизненным циклом карты

Активация и блокировка (`updateCard`)

1. **Проверка прав**: Только администраторы
2. **Поиск карты**: Получение карты по ID
3. **Обновление статуса**:
   - `ACTIVATE` → `Card.Status.ACTIVE`
   - `BLOCK` → `Card.Status.BLOCKED`
4. **Транзакционное сохранение**: Атомарное обновление в БД

Планировщик истечения срока действия

- **Периодичность**: Каждые 24 часа автоматически
- **Логика**: Поиск активных карт с истекшим сроком действия
- **Действие**: Автоматическая смена статуса с `ACTIVE` на `EXPIRED`

Удаление карты (`delete`)

1. **Проверка прав администратора**: `@PreAuthorize("hasRole('ROLE_ADMIN')")`
2. **Получение данных карты**: Поиск по ID и расшифровка номера
3. **Каскадное удаление**:
   - Удаление HMAC-хеша из `CardHash`
   - Удаление зашифрованного ключа из `CardEncryptionKey`
   - Удаление заявок на блокировку из `CardBlockingRequest`
   - Удаление самой карты из `Card`
4. **Транзакционная целостность**: Все операции в одной транзакции

Отображение информации о картах

Маскированное отображение (`mapToCardInfoResponse`)

1. **Расшифровка номера**: Временная расшифровка для маскирования
2. **Применение маски**: Формат `**** **** **** XXXX` (последние 4 цифры)
3. **Форматирование даты**: Срок действия в формате `MM/yy`
4. **Сборка ответа**: `CardInfoResponse` с безопасными данными

Полная информация для администраторов (`getAllCards`)

- **Права доступа**: Только администраторы
- **Полная расшифровка**: Показ реальных номеров карт
- **Тестовое назначение**: Метод для отладки и тестирования

Валидация и проверки безопасности

Проверка доступности карты (`checkCardAvailable`)

1. **Статус карты**: Должен быть `ACTIVE`
2. **Срок действия**: `!card.getValidityPeriod().isBefore(LocalDate.now())`
3. **Исключение**: `AccessDeniedException` при недоступности

Валидация операций с балансом (`validateBalanceUpdateCorrect`)

1. **Проверка переполнения**: `currentBalance > Double.MAX_VALUE - sum`
2. **Проверка отрицательного баланса**: `newBalance < 0`

3. **Исключения**: `IllegalArgumentException` с описанием ошибки

Контроль прав доступа

- **Извлечение аутентификации**: `SecurityContextHolder.getContext().getAuthentication()`
- **Проверка администратора**: Поиск authority `ROLE_ADMIN`
- **Контроль владельца**: Доступ только к собственным картам для обычных пользователей

Генерация уникальных идентификаторов

Алгоритм генерации номера (`generateUniqueCardNumber`)

1. **Цикл генерации**: До 100 попыток создания уникального номера
2. **Формирование номера**:

   java

   ```
   for (int i = 0; i < 4; i++) { int randomNumber = ThreadLocalRandom.current().nextInt(0, 10000); resultNumber.append(String.format("%04d", randomNumber)); }
   ```
3. **Проверка уникальности**: Вычисление и проверка HMAC-хеша
4. **Сохранение хеша**: Добавление в `CardHash` для предотвращения дубликатов
5. **Обработка неудачи**: `RuntimeException` при исчерпании попыток

Управление сроком действия (`setValidityPeriod`)

- **Значение по умолчанию**: 24 месяца (настраивается через `card.months-until-expires`)
- **Вычисление даты**: Последний день месяца через N месяцев от текущей даты
- **Формула**: `today.plusMonths(months).with(TemporalAdjusters.lastDayOfMonth())`

Пагинация и фильтрация

Получение списка карт (`getCardsInfo`)

1. **Определение владельца**:
   - Администратор может просматривать карты любого пользователя
   - Обычный пользователь видит только свои карты
2. **Применение фильтров**:

- По статусу карты (опционально)
- По ID владельца

3. **Пагинация**: `PageRequest.of(page, size)`
4. **Маппинг результатов**: Преобразование в `CardInfoResponse` с маскированными номерами

Обработка ошибок и исключений

Стратегия обработки ошибок

- **Маскирование деталей**: Общие сообщения "Internal error" для пользователей
- **Детальное логирование**: Полная информация об ошибках в логах системы
- **Типизированные исключения**:
  - `NotFoundException` - карта или пользователь не найден
  - `BadRequestException` - некорректные входные данные
  - `UnauthorizedException` - проблемы с аутентификацией
  - `AccessDeniedException` - недостаточно прав доступа

Транзакционная целостность

- **Аннотация** `@Transactional` : Все операции модификации данных
- **Откат изменений**: Автоматический rollback при исключениях
- **Атомарность операций**: Гарантия целостности при сложных операциях