Acuerdo.

Ethereum Payment DApp on Payment Channels

Зацепин В.В.

ФКН ВГУ

<https://github.com/zatsepinvl/acuerdo>

Воронеж 2018

Оглавление

[Введение 2](#_Toc530318529)

[Постановка задачи 2](#_Toc530318530)

[Анализ задачи 3](#_Toc530318531)

[Основные объекты системы 5](#_Toc530318532)

[Средства реализации 6](#_Toc530318533)

[Реализация 7](#_Toc530318534)

[Общее описание 7](#_Toc530318535)

[Запуск приложения 8](#_Toc530318536)

[Экраны пользовательского приложения 8](#_Toc530318537)

[Заключение 8](#_Toc530318538)

# Введение

Ethereum - блокчейн (blockchain) платформа, для построения распределенной сети или использования публичных: mainnet, ropsten, rinkeby и т.д. Ethereum Mainnet (главная сеть) можно считать платежной системой, основанной на внутренней валюте - токенах. Базовый токен для совершения всех операций и платой за транзакции майнерам - ETH (1 ETH ~  200$ на момент написания). Кроме внутренней валюты, Ethereum позволяет публиковать так называемые умные контракты (smart contracts) для выполнения бизнес логики, выполняющиеся на EVM (Ethereum Virtual Machine. Таким образом, на базе технологии Ethereum можно строить различные сервисы, которые будут обладать основными свойствами блокчейна: распределенность, безопасность, финализация. Однако дизайн системы блокчейна как платежной системы ограничивает пропускную способность всей сети до 15 транзакций в секунду, что не может сравниться, например с VISA, у которой этот показатель достигает 2000 транзакций в секунду. В результате, для того, чтобы совершить обычную операцию платежа за товар или услугу необходимо ждать около 1-2 минут. Кроме того, каждая транзакция требует оплаты комиссии, поэтому микроплатежи является и вовсе невыгодными. Для решения этих проблем существует два основных направления: L1 и L2 Scaling Solutions. L1 (Level 1) решения основаны на изменении самого протокола работы Ethereum или создания подсетей, например, технология Plasma. L2 (Level 2) решения основаны на создании специальных умных контрактов внутри существующей сети, которые не требуют изменений протокола. Одним из таких L2 подходов является Payment Channel.

# Постановка задачи

Необходимо разработать децентрализованное веб-приложение (DApp) для совершения платежей между двумя пользователями на основе Payment Channel Scaling Solution.

# Анализ задачи

В основе решения на Payment Channel лежит сущность канала (channel). Канал представляет собой направленное движение средств от одной стороны сделки к другой. На рисунке 1 представлена общая схема работы канала.

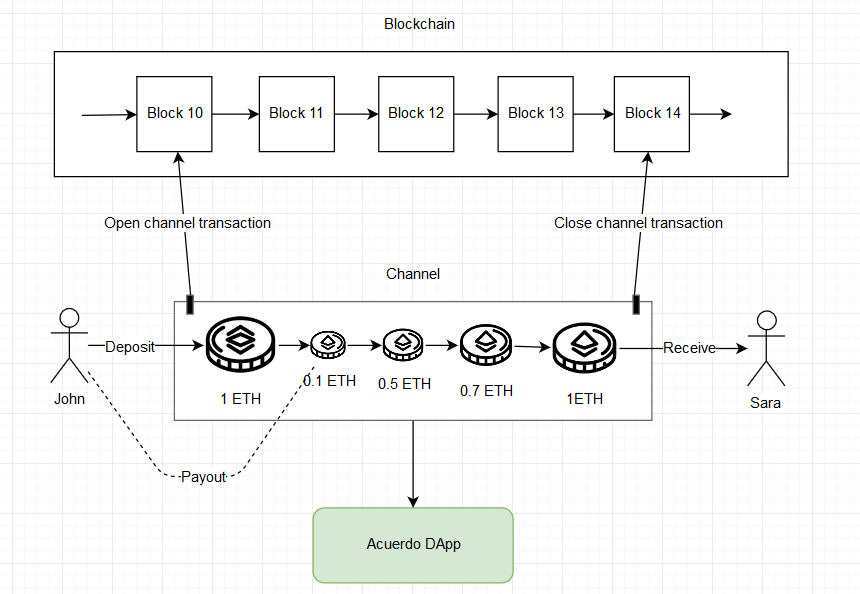


Рисунок 1 – схема работы Payment Channel

Для создания канала отправителю платежей (John) необходимо создать его и отправить средства, это действие совершается одной транзакцией в блокчейн. Далеу внутри канала происходит выплата средств получателю (Sara), транзакция и оплата ее комиссии для этого не требуется. Выплата происходит на основе подписания платежа отправителем и отправкой этой подписи получателю, таким образом, что он может проверить, что отправитель действительно подписал платеж. Схема работы отправки платежа представлена на рисунке 2.

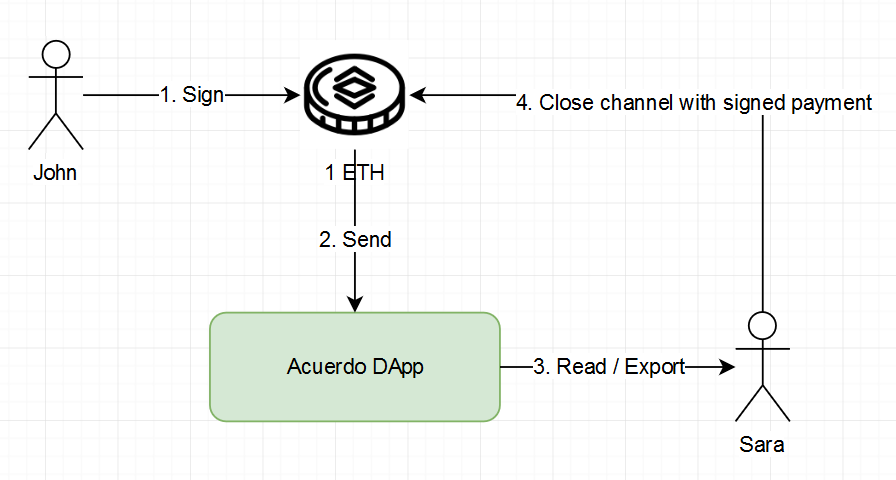


Рисунок 2 – схема отправки и применения платежа

Для того, чтобы получатель не запоминал сам все платежи, они проверяются и сохраняются внутри приложения Acuerdo. Подпись и проверка подлинности подписания осуществляется на основе приватного ключа пользователя и алгоритма цифровой подписи ECDSA, поддержка которого встроена в Ethereum.

После совершения платежа отправителем, возможно закрытие канала получателем и вывод средств. Каналы дают возможность совершать неограниченное количество микроплатежей, благодаря чему могут удовлетворить требованиям большинства операций с перевод средств в реальной жизни. Таким образом выделяются варианты использования системы, представленные на рисунке 3.

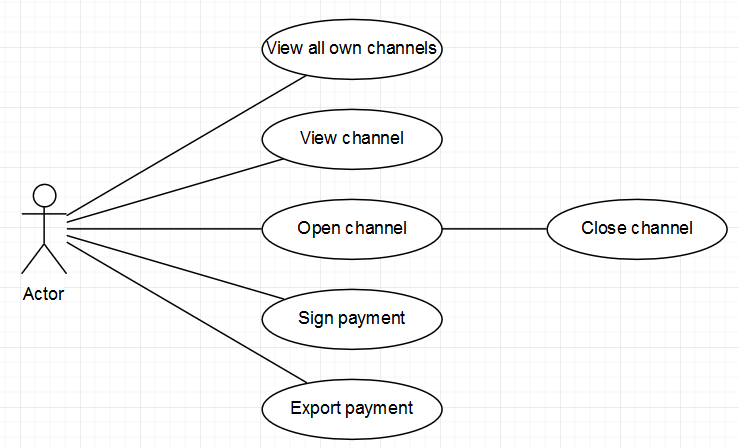


Рисунок 3 – варианты использования приложения

## Основные объекты системы

Объекты системы могут быть сохранены в смарт контракте (on-chain) или те, которые вынесены за его приделы (off-chain) по причине отсутствия надобности для операционной логики. Данные смарт контракта определяют его текущее состояние, транзакция в блокчейне, которая может является вызовом метода смарт контракта, переводит контракт в другое состояние. Открытие и закрытие канала – транзакции, которые изменяют состояние контракта. Подписание и отправка платежа выполняются за пределами блокчейна, поэтому не являются часть состояния контракта. Таким образом объекты системы можно разделить следующим образом:

* On-chain:
  + Channel – сущность канала, которая определяется как:
    - ChannelId – идентификационный номер канала
    - Sender – отправитель, определяется адресом в сети Ethereum
    - Recipient – получатель, определяется адресом в сети Ethereum
    - Amount – количество средств, которые могут быть выплачены получателю отправителем.
    - OpenedAt, ClosedAt – даты открытия и закрытия канала.
    - Duration – количество времени, когда средства, отправленные в канал блокируются только на операцию закрытия канала по определенному платежу. Если пои стечению этого времени канал не был закрыт, отправитель может сделать это, получив все средства назад. Данное время необходимо для того, чтобы избежать ситуаций, когда вторая сторона не отвечает.
* Off-chain:
  + Payment – сущность платежа, которая определяется как:
    - PaymentId – идентификационных номер платежа
    - ChannelId – идентификационный номер канала, к которому относится платеж
    - Amount – сумма платежа
    - SenderSignature – подпись отправителя
    - RecipientSignature – подпись получателя
    - CreatedAt – время выполнения платежа

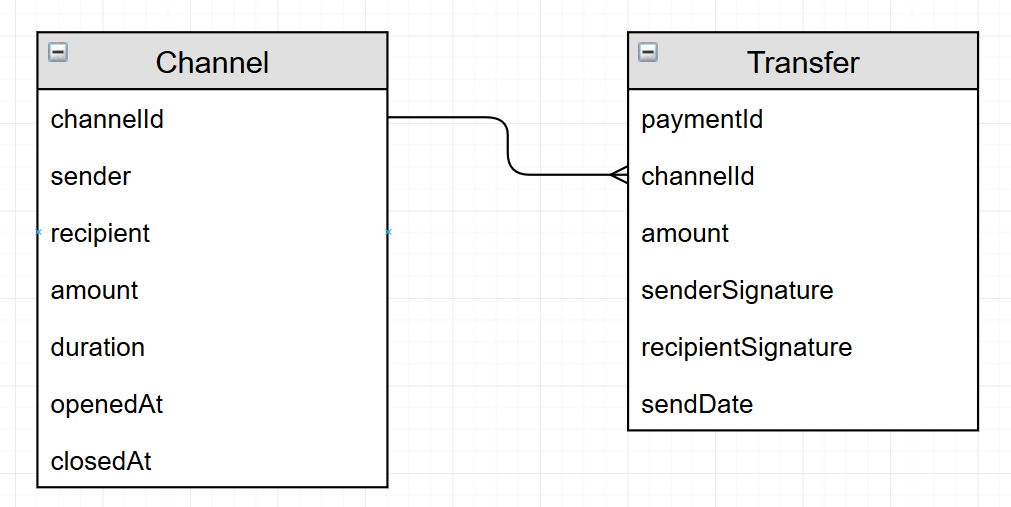


Рисунок 4 – ER диаграмма

## Средства реализации

Для реализации приложения были выбраны следующие средства:

* Blockchain
  + Ethereum – блокчейн платформа
  + Solidity – язык написания смарт контракта
  + Truffle – js фреймворк для компиляцит, разворачивания и тестирования смарт контрактов
  + Ganache-cli – тестовая частная Ethereum сеть
* Server
  + Java
  + Kotlin
  + Web3j – библиотека для работы с Ethrereum
  + Spring Boot
  + Gradle
  + MongoDB
  + JWT
* Webapp
  + ECMAScript 6 (JavaScript)
  + ReactJS
  + NPM
  + Webpack
  + Mobx – фреймворк для управления состоянием клиентского приложения
  + MaterialUI – набор React компонентов для построения пользовательского интерфейса согласно Google Material UI Guideline
  + Web3js - библиотека для работы с Ethrereum
  + MetaMask – плагин для браузера, через который пользователь взаимодействует с блокченом
* Развертывание и контейнеризация
  + Docker
  + Docker-compose

# Реализация

## Общее описание

Общая архитектура приложения представлена на рисунке 5.

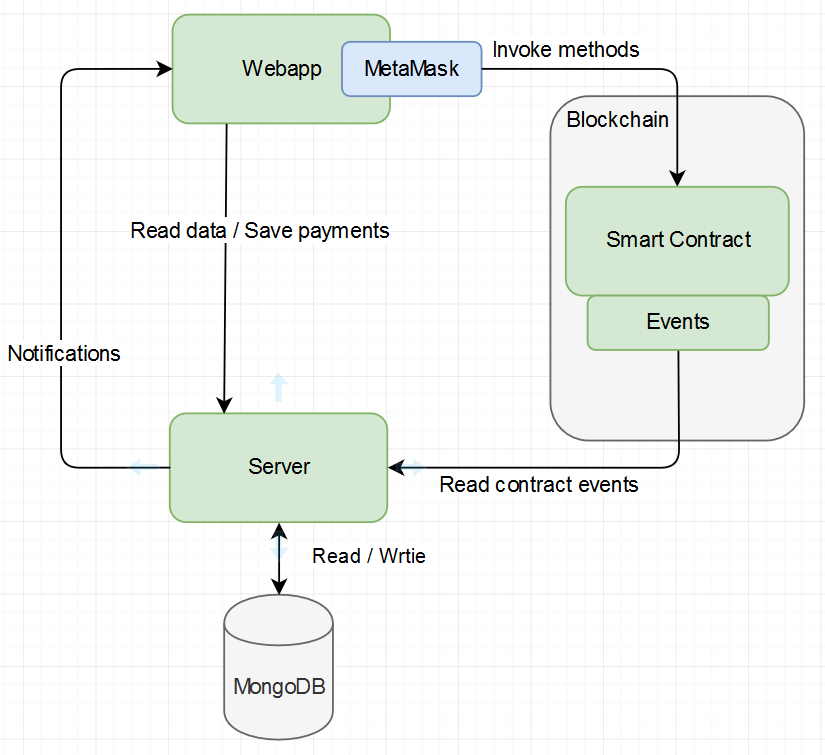


Рисунок 5 – общая архитектура приложения

Пользователь взаимодействует с Webapp приложением и через MetaMask плагин совершает действия в блокчейне. Каждая операция создания и закрытия канала в блокчейне запоминается в логах контракта, которые периодически читает server и сохраняет копию данных в MongoDB. Это необходимо для того, чтобы пользователи имели быстрый доступ к своим каналам. Кроме того, server отвечает за то, чтобы сохранять платежи и отправлять оповещения на клиент по веб-сокетам.

## Создание канала

Процесс создания канала представлен на рисунке 6.

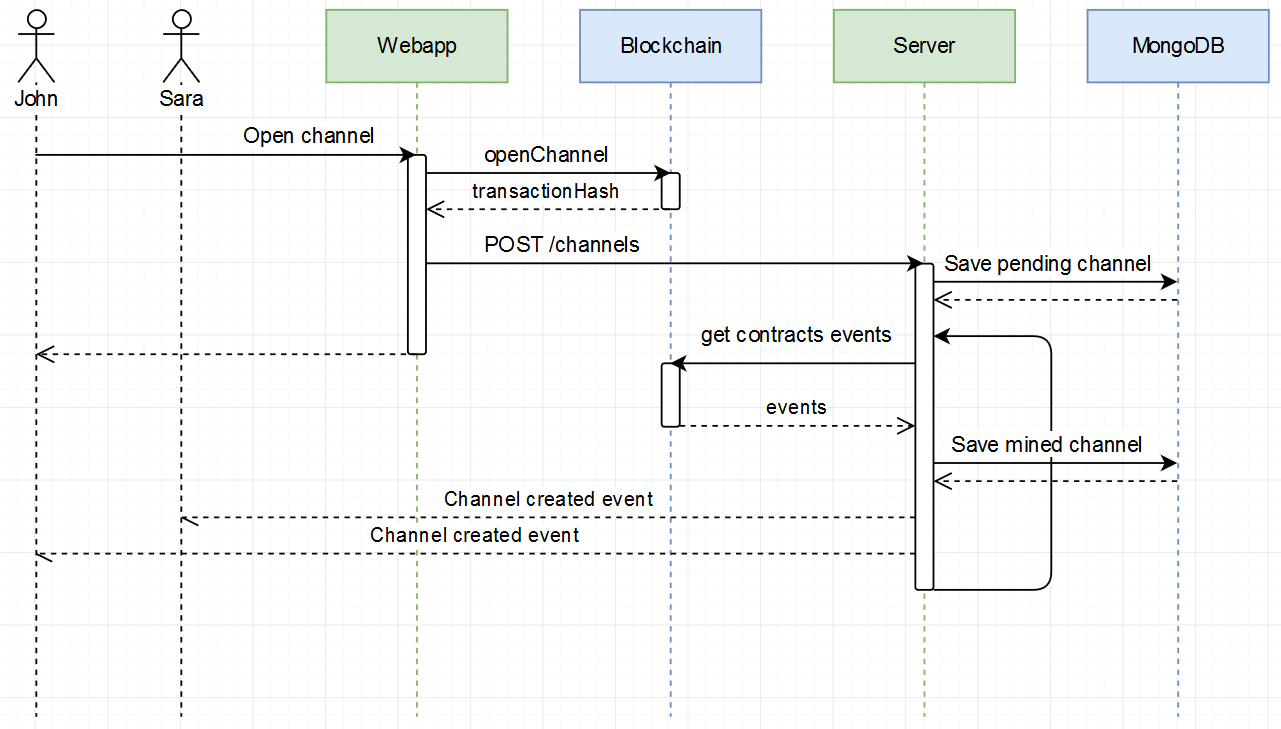


Рисунок 6 – процесс создания канала

John (отправитель) создает канал, отправляя транзакцию на в блокчейн, после чего получает ее хэш (транзакция отправлена, но еще не замайнена). Далее клиентское приложение отправляет POST запрос на сервер, чтобы сохранить там данные о том, что ожидается событие открытие канала (статус канала Pending). В свою очередь сервер периодически опрашивает blockchain на предмет новых событий, и как только получает их, сохраняет открытие каналы в базу данных и оповещает об этом клиента по веб-сокетам.

## Аутентификация

Данные в Ethereum блокчейне являются публичными по дизайну, т.е. информация о всех транзакциях, выполнениях методов смарт контрактов, отправителях и получателях является общедоступной. Это свойство распространяется и на сущность канала, которая является on-chain. «Чувствительных» для пользователя данных сервер не хранит, поэтому явной необходимости для обеспечения безопасности данных нет. Тем не менее, явная аутентификация пользователей дает повышенный уровень безопасности и исключает простой доступ к данным пользователей. Процесс аутентификации (рисунок 7) выстроен на подписи определенного сообщения (та же цифровая подпись ECDSA что я для отправки платежей) через MetaMask и отправки запроса на сервер, которые проверяет правильность подписи и выставляет JWT Cookie, использующийся при следующих запросов с этого же клиента.

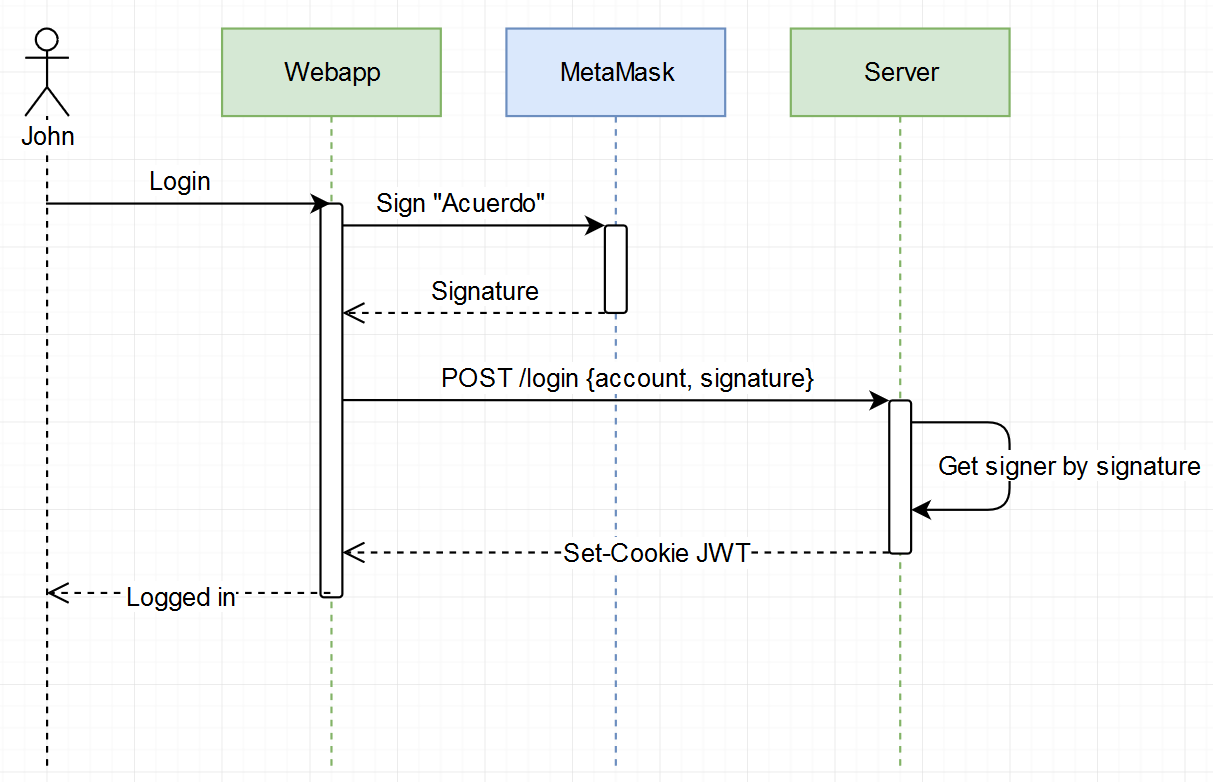


Рисунок 7 – процесс аутентификации

## Запуск приложения

## Экраны пользовательского приложения

# Заключение