**1. Introduction**

This document intends to describe the smart contract of the Frizbee project, that will help to understand the product flow, technical implementation and to be a starting point for the testing process.

**2. Project Structure**

For implementation it is used the Truffle Framework v4.1.13 (<https://truffleframework.com/>).

*| -* ***build***

*| -* ***config***

*| -* ***contracts***

*| - AbstractPaymentQuantum.sol*

*| - Quantum.sol*

*| - Migrations.*

*| -* ***coverage***

*| -* ***migrations***

*| -* ***scripts***

*| -* ***test***

*| - truffle.js*

**3. Keywords**

* ***Platform*** – the Frizbee platform
* ***Platform Wallet*** – the Frizbee platform wallet
* ***Platform Fee*** – the amount of wei the Platform collects after each video sessions
* ***Platform Rate*** – the rate that is used to calculate the Platform Fee
* ***Knowledge Provider*** (KP) – the creator and owner of a video session
* ***Knowledge Provider Rate*** – the KP’s hour based rate per video sessions
* ***Participant*** – the video session participants
* ***Payment Pool*** – the bridge service between the application and smart contract
* ***Frizbee Wallet*** – the wallet generated at the application sign-up

**4. Functional Flows**

**4.1 Deposit funds**

In order to participate to Frizbee video sessions, each participant needs to deposit funds in the smart contract. The funds are transferred by the user from the Frizbee wallet and stored in the smart contract. The smart function is defined in AbstractPaymentQuantum class and is implemented in the base contract class that extends this abstract class.

//AbstractPaymentQuantum.sol

function deposit() payable public returns (uint);

**//**Quantum.sol

function deposit() public payable returns (uint) {

require(withdrawMap[msg.sender].withdrawed == 0);

...

}

Important to note the require() condition that will be detailed below.***(\*)***

**4.2 Withdraw funds**

Withdraw action is the reverse of deposit action and clears user’s deposit by moving all the available funds from the contract storage to the Frizbee Wallet. Between the withdraw and deposit actions there is a conditional requirement ***(\*)*** that prevents the deposit of new funds in the contract as long as there is a withdraw in progress.

The withdraw process consists of two parts:

* initialization
* withdraw

The initialization action marks a flag in the contract storage that indicates that a withdraw is in progress for the specific user owner of the deposit. At payment channel level, a new record is saved in the payment pool with relevant data such as a cryptographically signed message, or a cooldown timer.

**//**Quantum.sol

function initWithdraw()

public

hasFunds(msg.sender)

returns (bool)

{

if (withdrawMap[msg.sender].withdrawed == 0) {

withdrawMap[msg.sender].withdrawed = 1;

withdrawMap[msg.sender].time = now;

}

return true;

}

The withdraw action clears the deposit, as we said before. The withdraw action can be done in two ways.

* autowithdraw
* direct withdraw

The autowithdraw consists in an autorun service checks all the withdraw initiated records and if the cooldown period has passed, a transaction is sent to the Smart Contract, along with the signed message. Validation is done with the ecrecover method. If validation successes, the deposit is processed.

**//**Quantum.sol

function autoWithdraw(address \_participant, uint8 \_v, bytes32 \_r, bytes32 \_s)

public

hasFunds(\_participant)

returns (bool)

{

if (\_participant != ecrecover(keccak256(\_participant), \_v, \_r, \_s)) {

revert();

}

bool success = false;

if (now > withdrawMap[\_participant].time + withdrawTimeLimit &&

withdrawMap[\_participant].withdrawed == 1) {

if (consumeDeposit(\_participant)) {

resetWithdraw(\_participant);

success = true;

}

}

return success;

}

If for some reason the autowithdraw process fails, the fallback action is the direct withdraw which can be triggered by the user from the application.

**//**Quantum.sol

function withdraw()

public

hasFunds(msg.sender)

returns (bool)

{

…

A note here about the withdraw time limit element, which is a time buffer that delays the withdraw action to allow payments to be made from the deposit.

**4.3 Commit payments**

The fee that needs to be paid by each participant is updated in real time during the video session at a regular time interval (10 seconds). Relevant information, including a cryptographically signed message, are stored in thepayment pool.

Once the video session is ended by the KP, or by the participant (by leaving the video session), the equivalent record from the payment pool is marked as ready to be committed to the Smart Contract. An autorun service checks the payment pool for ready-to-commit marked records and a transaction will be sent to the Smart Contract along with the signed message. At the Smart Contract level, the signed message is validated by the ecrecover method. If the validation succeeds, the payments are processed.

**//**Quantum.sol

function commit(

address \_participant,

address \_kp,

uint \_value,

uint \_sessionId,

uint8 \_v,

bytes32 \_r,

bytes32 \_s

)

public

isValidAmount(\_participant, \_value, \_sessionId)

returns (bool)

{

if (\_participant != ecrecover(keccak256(\_sessionId, \_participant, \_kp, \_value), \_v, \_r, \_s)) {

revert();

}

…..

remainingValue[0] -= collectPlatformFee(remainingValue[0]);

emit Commit(\_participant, \_kp, \_value, remainingValue[0], \_sessionId, now);

return payKp(\_kp, remainingValue[0]);

}

The methods collectPlatformFee() and payKp() take as parameter the value commited by the participant for a video session and by applying the Platform Rate it calculates the platform fee and adds it to the Platform Wallet, for the first function, and the rest of the amount is sent to the KP’s Frizbee Wallet.

**4.4 Transfer to platform the collected fees**

To collect the platform fees, an autorun services that will run probably once a day will send all the fees collected during this period (with the collectPlatformFee() method).

//Quantum.sol

function payPlatform() public

onlyBy(platform)

returns (uint)

{

uint fee = platformFee[wallet];

platformFee[wallet] = 0;

wallet.transfer(fee);

emit Payment(wallet, fee, now);

return platformFee[wallet];

}