

Multi-step payment tracking dapp “Smart deal”

Problem

Multi-step business deals, agile projects, complex agreements have a common issue of payment tracking and fulfillment. Usually a 3rd party has to be involved to control if deliverables are completed and an agreed sum of money for each stage is paid in time.

Solution

“Smart deal” is a convenient and reliable decentralized system that helps to define monetary agreement between a client and an agent. Moreover, It helps to prevent deals with unqualified agents by storing history of completed deals, reviews and ratings.

"Smart deal" solution includes a convenient web-based portal with authorized consultants that help to create an agreement and resolve all the issues.

As a starting point, a client comes up with specific requirements for a task and picks an agent by looking at the rating and reviews. Then the client and the agent get connected via a web-based platform to discuss the requirements and every task of the deal. Each task section contains a task name and a short description. The agent has to acknowledge himself with the deal and evaluate every task. By the end of the negotiation process the contract gets submitted to the system where authorized consultants register it in a blockchain.

Once the contract gets registered:

1. Both parties transfer the agreed percentage of the original estimated money to the contract address for protection purposes (read below).
2. Deal changes its status to active
3. First task gets opened
4. Client's money for the first task get reserved by transferring it to contract's address
5. Agent works on the pending task.

When the task is done:

1. Agent marks the task as “ready for review” in the system and sends evidence to the client for the verification
2. If the client is happy with the results, he/she accepts the task in the system. Next task gets opened.
Otherwise the task gets rejected and returned back to the agent with additional comments (which are sent within a portal).
3. Cycle repeats with the new task.

At the end of the deal, when the last task is completed, a deal comes to its logical end. The client leaves a rating and a review about the agent's work. Protection money gets transferred back from the contract's address to contract parties.

As a result, both parties complete the agreement with no extra cost and worries for the project and transaction management. Moreover, there is a special protection mechanism that

ensures that one party gets compensated in case of another one dropping the contract. Compensation is calculated based on the percentage of the contract's original estimated value and is obtained during the contract creation.

Solution is published on [github](#).

Blockchain details

Public blockchain

"Smart deal" is deployed on a public blockchain. Public blockchain is the most transparent type of blockchain where anyone is free to join and participate in the core activities! It also encourages new users to join the network. It is a win-win solution for both parties - agents are motivated to keep their work on a decent level to get positive reviews and ratings, which are stored forever inside a blockchain, while clients can browse through agents and their history of all the deals and pick the most highly-rated and suitable agent for their particular tasks.

Storing the data

Blockchain is not meant to store huge documents, images or texts as it is costly and time-consuming, therefore there are two ways to overcome that limitation:

1. Use ready blockchain solutions for storing data in a decentralized way via dedicated protocol.
2. Store only necessary information in a blockchain, while providing detailed info in a web-based portal.

As connecting a prototype solution to a real working blockchain solution by a 3rd party is beyond the requirements of the course, a second approach was picked.

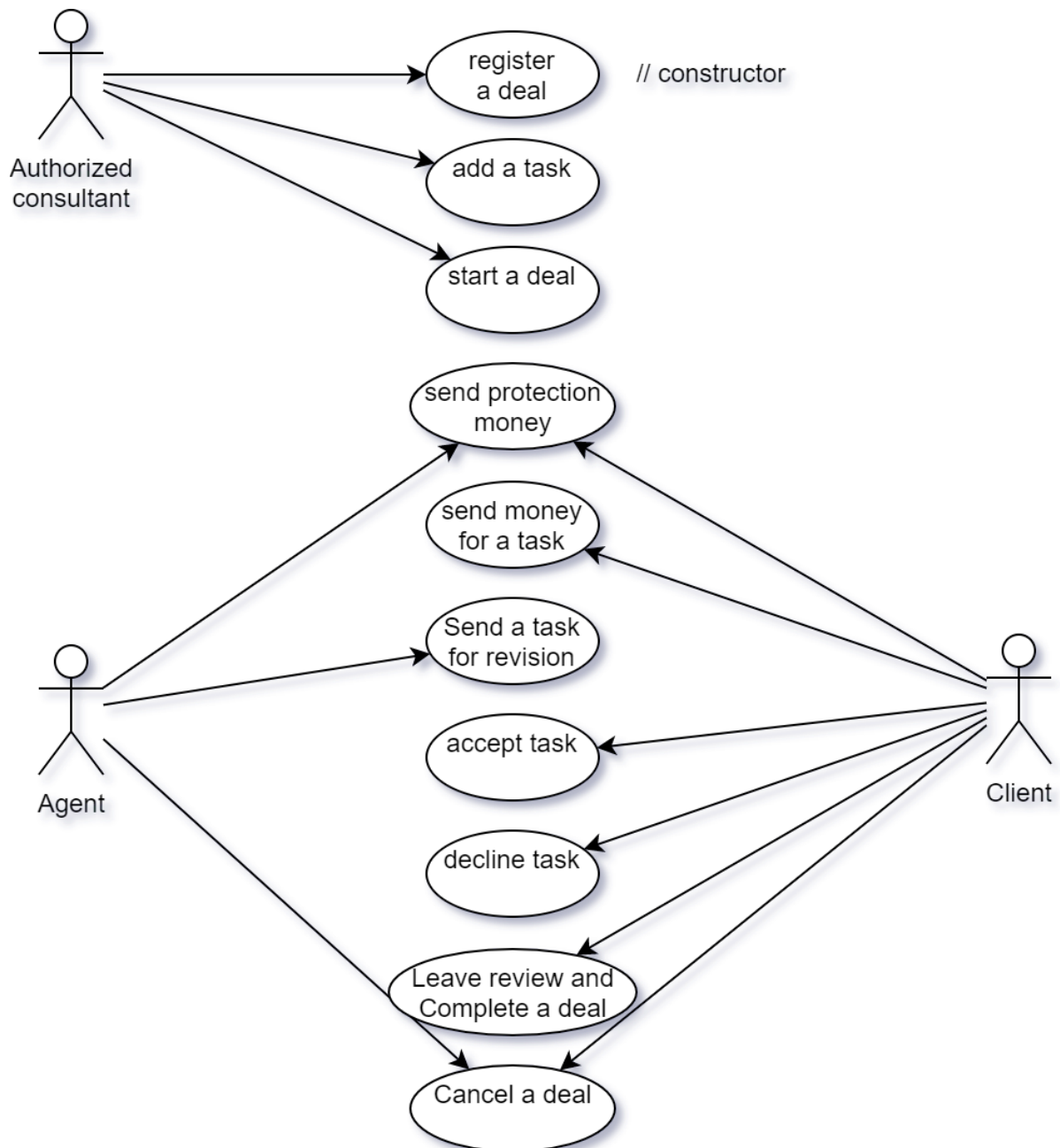
In the scope of the "Smart deal", a web-based portal should provide a way for a customer to communicate with an agent. Portal will handle the following actions:

- Sending a deal
- Negotiating about the prices
- Sending revision notes

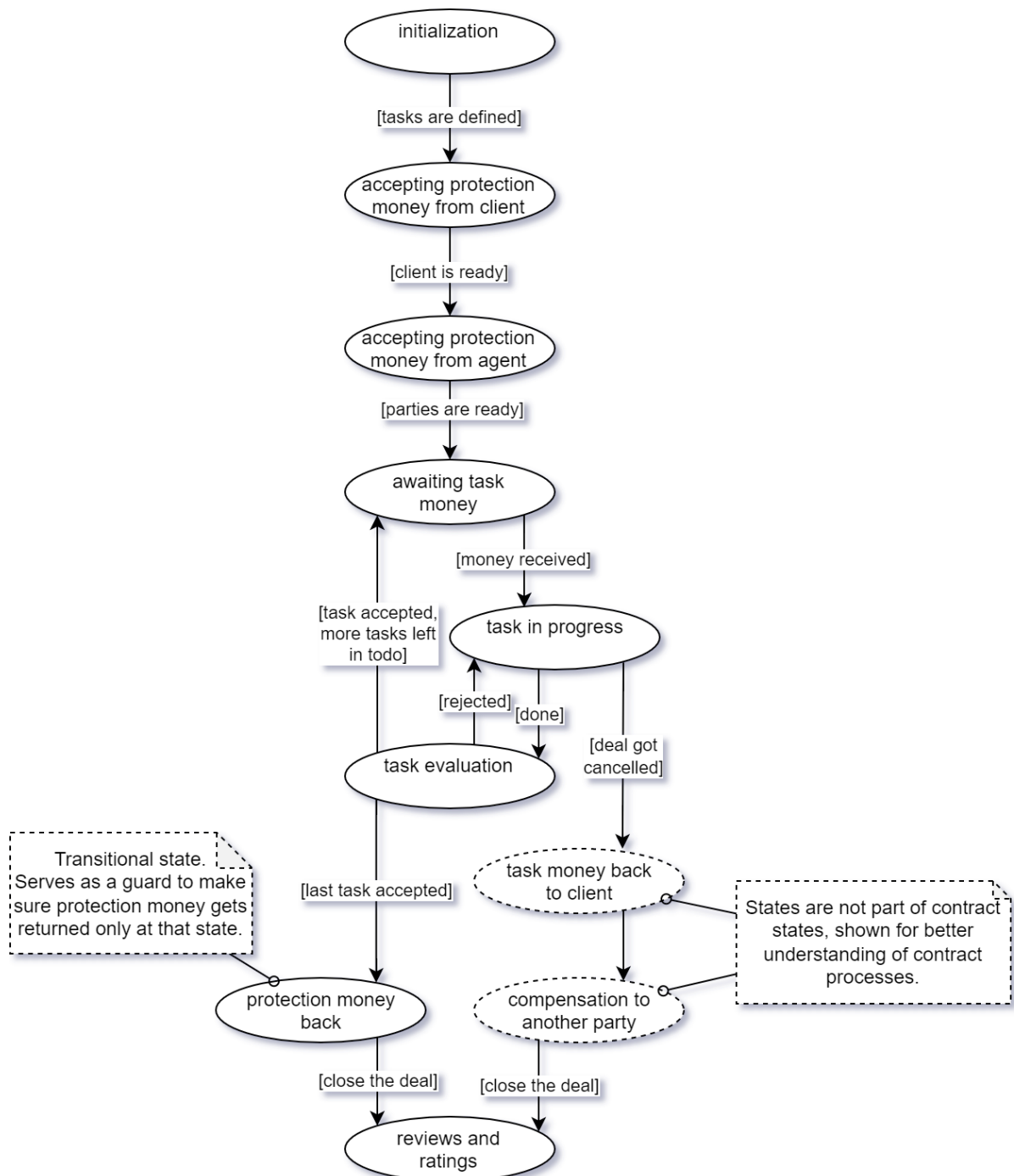
The following transactions will be recorded in a blockchain:

- Deal registration (initiator: authorized consultant)
- Sending protection money to contract's address (initiator: agent, customer)
- Sending money for a task (initiator: customer)
- Task sending for revision (initiator: agent)
- Task completion / rejection (initiator: customer)
- Deal's cancellation (initiator: agent, customer)
- Deal's completion (initiator: customer)

Use case diagram



Smart contract's state diagram



Detailed class diagram - fields, events, modifiers, functions

```

contract SmartDeal {
    struct Task {
        string title;
        uint amount;
    }
    enum State { Init, ProtectionFromClient, ProtectionFromAgent, PayForTask, TaskInProgress,
                TaskEvaluation, ProtectionMoneyBack, ReviewsAndRatings, Done }

    State public state = State.Init;
    Task[] public tasks;
    uint public taskIdx = 0;
    uint public contractValue;
    address public creator;
    address payable public agent;
    address payable public client;
    uint public protectionPercent;

    // ----- events

    event DealStarted();
    event DealCancelled(address initiator);
    event TaskAccepted(string title);
    event TaskDeclined(string title, string revisionMessage);
    event RatingLeft(uint rating, string review);
    event DealCompleted();

    // ----- Constructor + modifiers

    constructor(address _agent, address _client, uint _protectionPercent) {}

    modifier atState(State _state) {}
    modifier onlyAgent {}
    modifier onlyClient {}
    modifier onlyParties {}
    modifier onlyCreator {}

    // ----- utility functions

    function getProtectionValue() public view returns (uint money) {}
    function getCurrentTask() public view returns (Task memory task) {}
    function getBalance() public view returns (uint) {}
    function getTasks() external view returns (Task[] memory allTasks) {}

    // ----- logic function

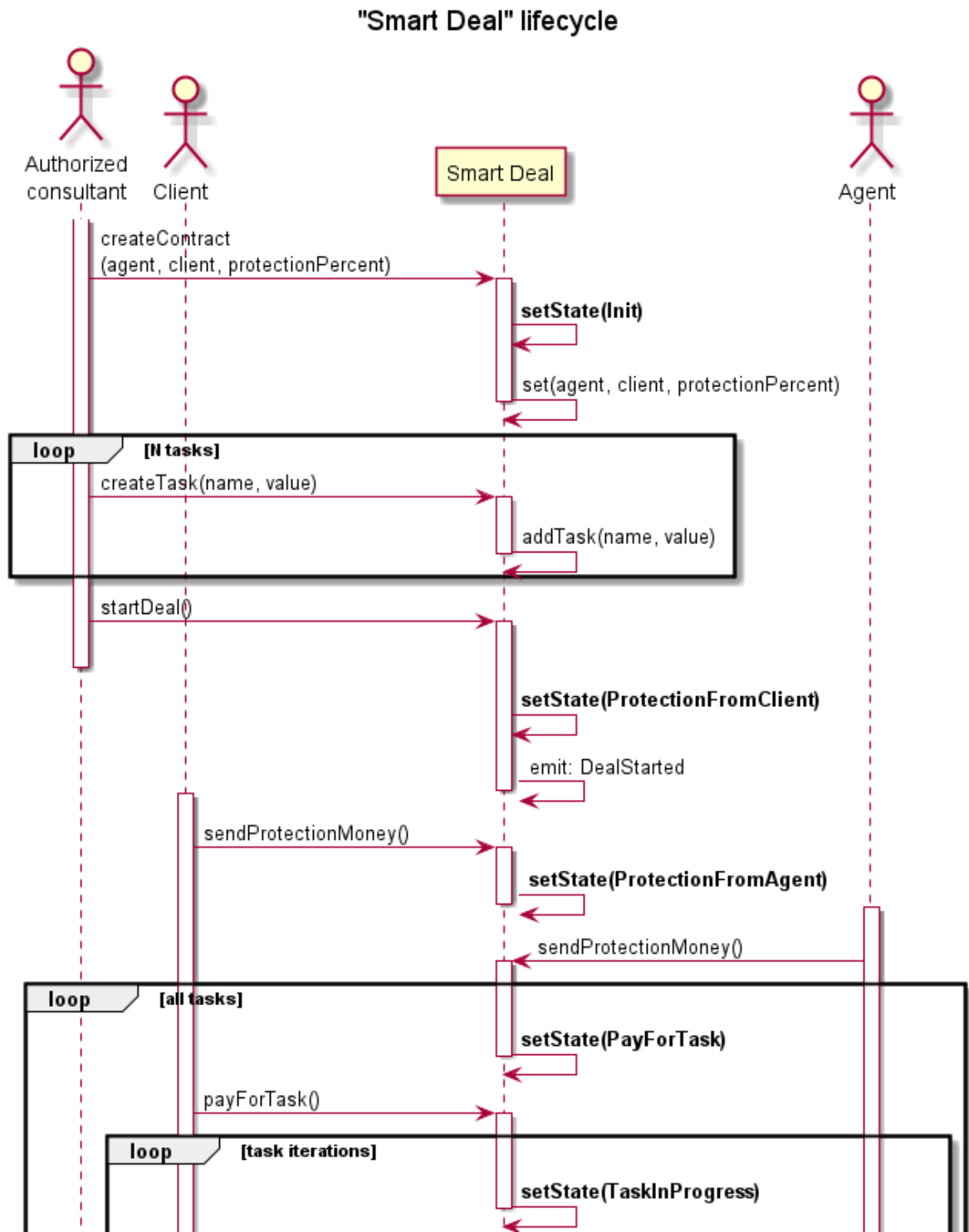
    function addTask(string memory title, uint amount) public atState(State.Init) onlyCreator {}
    function startDeal() public atState(State.Init) onlyCreator {}
    function sendProtectionMoney() public payable onlyParties {}
    function payForTask() public payable atState(State.PayForTask) onlyClient {}
    function sendTaskToRevision() public atState(State.TaskInProgress) onlyAgent {}
    function cancelDeal() public atState(State.TaskInProgress) onlyParties {}
    function acceptTask() public atState(State.TaskEvaluation) onlyClient {}
    function declineTask(string memory revisionMessage) public atState(State.TaskEvaluation) onlyClient {}
    function returnProtectionMoney() internal atState(State.ProtectionMoneyBack) onlyParties {}
    function writeReview(uint rating, string memory review) public atState(State.ReviewsAndRatings) onlyClient {}
}

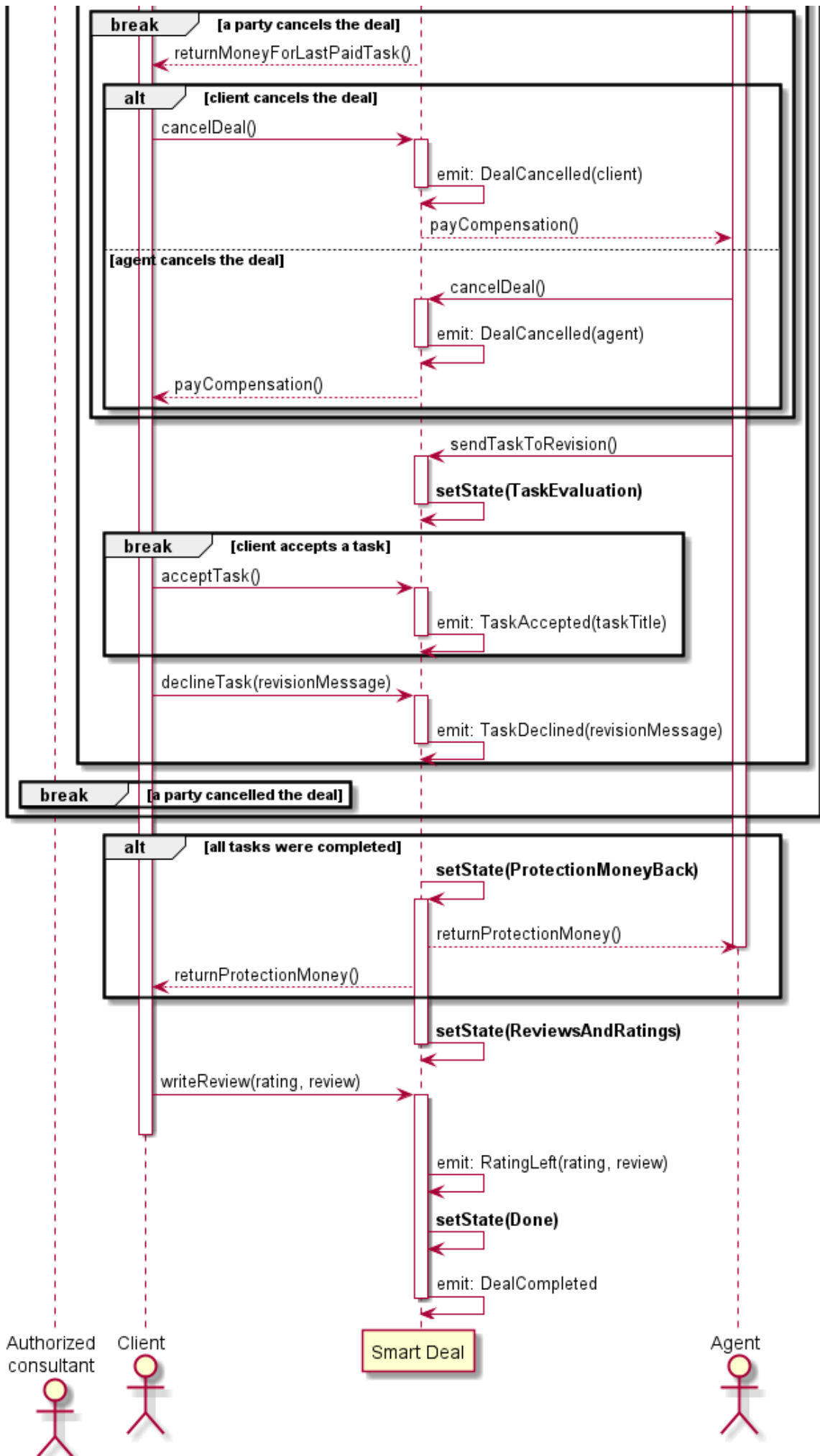
```

Sequence diagram

Note: Diagram does not represent error handling, it serves as an overview of the smart contract interaction.

Check attachments for full diagram without splits.





Deployed contract in Remix

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT

JavaScript VM (London)

ACCOUNT

0x4B2...C02db [97 1999999999971484]

GAS LIMIT

3000000

VALUE

0 Wei

CONTRACT

SmartDeal - contracts/SmartDeal.sol

DEPLOY

_AGENT: 0xA68483F649C6d1cf9B849A677d01

_CLIENT: 0x4B20993Bc481177e7E9C71eCaE8A

_PROTECTIONPERCENT: 30

Transact

☐ Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded 12

Deployed Contracts

▼ SMARTDEAL AT 0xD91...39138 (MEMORY)

acceptTask

addTask "two tasks","50000000000000000000"

cancelDeal

declineTask string revisionMessage

payForTask

sendProtectionM...

sendTaskToRevis...

```

1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity >=0.7.0 <0.9.0;
4
5 contract SmartDeal {
6     struct Task {
7         string title;
8         uint amount;
9     }
10    enum State { Init, ProtectionFromClient, ProtectionFromAgent, PayforTask, TaskInProgress, TaskEvaluation, ProtectionMoneyBack,
11
12    State public state = State.Init;
13    Task[] public tasks;
14    uint public taskId = 0;
15    // value for all the tasks
16    uint public contractValue;
17    // Can be initiated, but exposed for sake of displaying it in UI
18    address public creator;
19    address payable public agent;
20    address payable public client;
21    uint public protectionPercent;
22
23    // ----- events
24
25    event DealStarted();
26    event DealCancelled(address initiator);
27    event TaskAccepted(string title);
28    event TaskDeclined(string title, string revisionMessage);
29    event RatingLeft(uint rating, string review);
30
31    // ----- Constructor + modifiers
32
33    constructor(address _agent, address _client, uint _protectionPercent) {
34        require(_protectionPercent > 0 && _protectionPercent < 100, "Protection percent should be in range 1%-99% of the entire contr
35        creator = msg.sender;
36        agent = payable(_agent);
37        client = payable(_client);
38        protectionPercent = _protectionPercent;
39    }
40
41    modifier atState(State state) {

```

🔊 ☐ Listen on network 🔍 Search with transaction hash or address

📡 [call] from: 0xA68483F649C6d1cf9B849A677d01315835cb2 to: SmartDeal.state() data: 0xc19...d93fb

transact to SmartDeal.sendTaskToRevision pending ...

✅ [vm] from: 0xA68...35cb2 to: SmartDeal.sendTaskToRevision() 0xD91...39138 value: 0 wei data: 0x54d...1dc6 logs: 0 hash: 0xSec...34b95

transact to SmartDeal.acceptTask pending ...

✅ [vm] from: 0x4B2...C02db to: SmartDeal.acceptTask() 0xD91...39138 value: 0 wei data: 0x822...57623 logs: 1 hash: 0xb2a...f5fe4

SMARTDEAL AT 0XD91...39138 (MEMORY)

acceptTask

addTask "two tasks", "5000000000000000000"

cancelDeal

declineTask string revisionMessage

payForTask

sendProtectionM...

sendTaskToRevis...

startDeal

writeReview uint256 rating, string review

agent

client

contractValue

creator

getBalance

0: uint256: 4600000000000000000

getCurrentTask

getProtectionVaL...

0: uint256: money 1800000000000000000

getTasks

0: tuple(string,uint256[]): allTasks something,10000000000000000000000,two tasks,5000000000000000000000

protectionPercent

state

0: uint8: 4

taskidx

tasks uint256

Implementation details

Technologies

Git was used as a version control system. Solution can be viewed on [github](#).

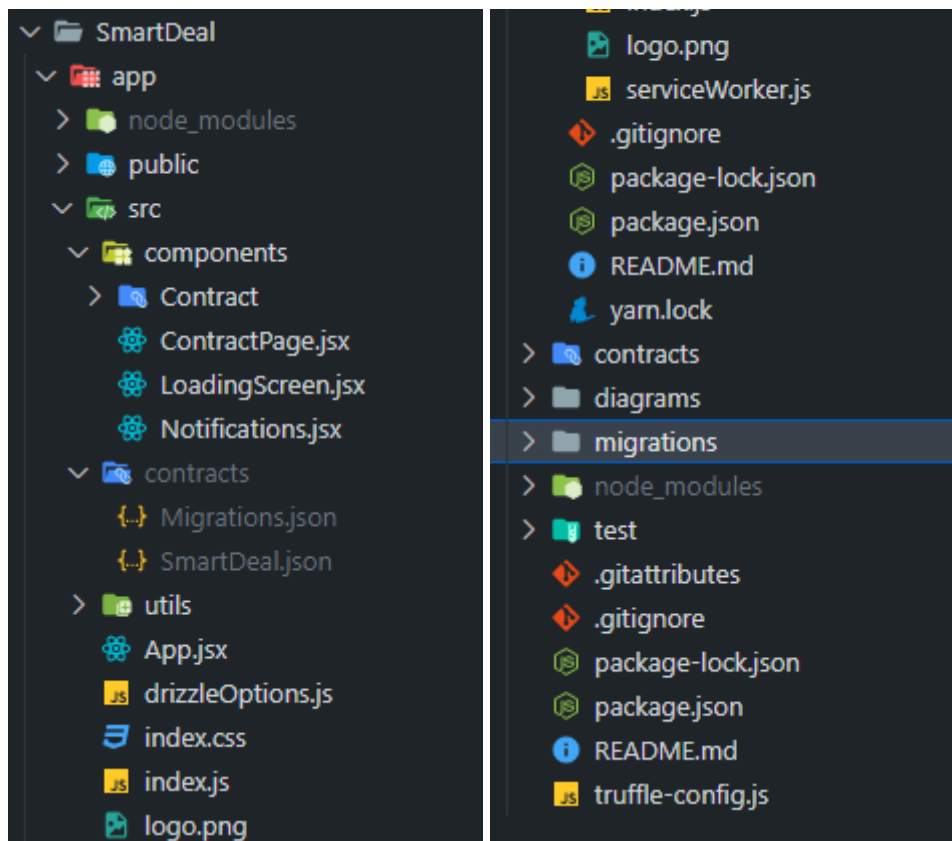
The entire [truffle suite](#) was used to develop and test Smart Contract (ganache + Truffle) and dapp (drizzle).

Tests for the Smart Contract were written using Mocha testing library, which was already integrated with the truffle suite.

Dapp was created with the [React](#) framework and drizzle on top of it, utilizing all provided drizzle modules (drizzle-react, drizzle-react-components, drizzle/react-plugin and drizzle/store).

As a design system [ant design](#) solution was picked that allowed use of existing components to build the interface and enhance user experience.

Folder structure



SmartDeal>app contains dapp frontend. Under **src>components** all react components are stored, including generic components like **Notifications.jsx** and contract-specific components under the **Contract** folder. **src>utils** contains useful utilities for components, including conversion from Wei to Eth and vice versa, all contract states in JS format and others. **src>contracts** folder contains all contracts in json format (generated by truffle following **SmartDeal>truffle-config.js** configuration). **src>drizzleOptions.js** is drizzle configuration. **src>index.js** is the application entry point.

SmartDeal>contracts has solidity contracts, while **SmartDeal>migrations** has necessary migrations.

SmartDeal>diagrams includes all the diagrams used in this document as well as diagram source files (sequence diagram was created via [plantUml](#) tool and source file contains all the code, while others were composed inside [draw.io](#) and contain according sources).

SmartDeal>test includes contract tests.

Tests

Using the [official truffle guide](#) for testing several unit tests were written for the contract.

Tests were created to represent how the contract can be tested. Tests do not cover all scenarios, just the main one with successful deal completion.

Tests can be run via **truffle test** command.

Note: in production unit tests should test each function separately. Once every function is covered, the entire contract can be tested in the whole.

```
PS C:\Users\Dmitrijs\uni\blockch\code\SmartDeal> truffle test
Using network 'test'.

Compiling your contracts...
=====
✓ Fetching solc version list from solc-bin. Attempt #1
✓ Fetching solc version list from solc-bin. Attempt #1
> Compiling .\contracts\Migrations.sol
> Compiling .\contracts\SmartDeal.sol
> Compiling .\contracts\SmartDeal.sol
> Artifacts written to C:\Users\Dmitrijs\AppData\Local\Temp\test--17820-6JqfjTuF34Sc
> Compiled successfully using:
  - solc: 0.8.11+commit.d7f03943.Emscripten.clang

Contract: SmartDeal
  ✓ should initialize with empty task array (940ms)
  ✓ should only allow creator to add tasks (2461ms)
  ✓ should successfully create a deal, complete all tasks and finish it (7241ms)

3 passing (11s)

PS C:\Users\Dmitrijs\uni\blockch\code\SmartDeal> 
```

Launch instructions

There are 2 possible ways to launch the dApp - by utilizing ganache's web3 provider or Metamask's one. The difference is that ganache's provider allows creating transactions without manual verification and provides access to all 10 (in default configuration) initially created accounts. Metamask, on the other hand, requires to verify each transaction by hand and gives access only to the currently selected address.

Launch instructions are described on [github](#).

Important note: during the initial contract deployment, ganache's 1st account's address is associated with the Authorized consultant role (contract creator), 2nd - with the Agent and 3rd - with the Client. Contract gets deployed with 30% of protection value.

DApp interface and interactions

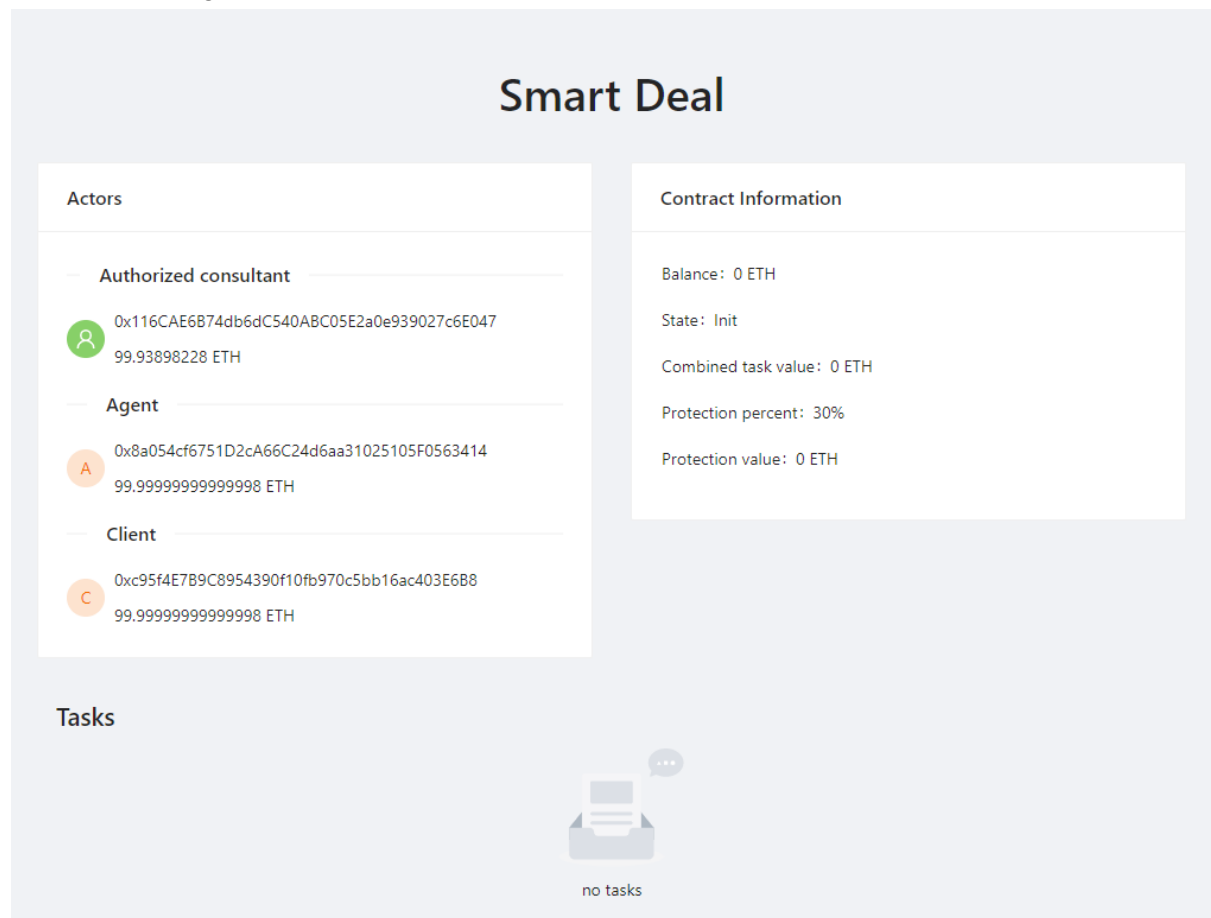
Description

The interface is clean, simple, responsive and follows best user experience practices. It is easy to navigate through and easy to learn!

For the given task only a contract page was developed. It Includes the following sections:

On the top of the page all the information about the contract is located, including Actors (creator - authorized consultant, agent and client), contract information, all contract tasks and available actions.

Below screenshots were taken during regular dApp launch utilizing only ganache's provider. When launching dApp via Metamask's provider, some components are simplified.



Tasks are followed by the section with all available actions (transactions). At the very top there is a dropdown with 3 actors mentioned above. By picking an actor you can specify who is going to send the chosen transaction.

There is a delimited with a state name before each transaction. It indicated at which state of the contract transaction can and should be performed.

Actions

Select a user:

Creator ▾

Creator
Agent
Client

Init

* name:

* amount (ETH):

Add task

Start deal

ProtectionFromClient, ProtectionFromAgent

Send protection money

PayForTask

Pay for task

TaskInProgress

Send task to revision

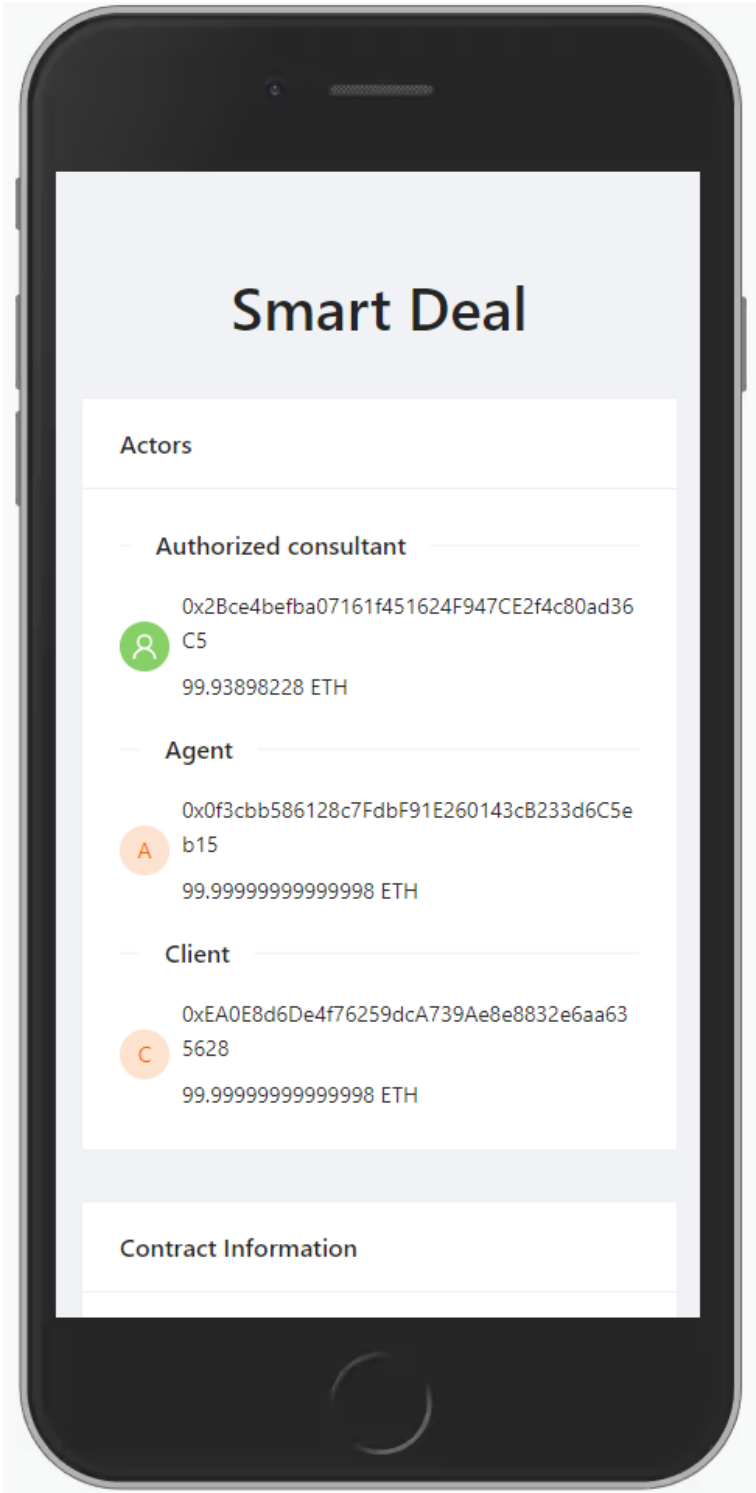
Cancel deal

Last section is dedicated to smart contract's emitted events.

Events

No Data

User interface is not limited to big screen devices only. It is responsive and supports also mobile devices



Interaction, scenario 1 - successfully finish the deal (ganache's provider)

Balance at the beginning. Keep in mind, that first address is associated with the Authorized consultant role (contract creator), 2nd - with the Agent and 3rd - with the Client.

ADDRESS	BALANCE
0x2Bce4befba07161f451624F947CE2f4c80ad36C5	99.94 ETH
0x0f3cbb586128c7FdbF91E260143cB233d6C5eb15	100.00 ETH
0xEA0E8d6De4f76259dcA739Ae8e8832e6aa635628	100.00 ETH

Let us create several tasks by using creator's account

Actions

Select a user: Creator ▾

Init

* name:

* amount (ETH):

Add task

Notifications on the right upper corner show information about transactions

i
transaction
×

0x2ef850093b4b142c31b177f91d09a
d87ebec231e0de7e887af657bee1b9
42669
Transaction is pending

✓
transaction
×

0x2ef850093b4b142c31b177f91d09a
d87ebec231e0de7e887af657bee1b9
42669
Transaction is successfull! Block number: 5, gas
used: 108085

Let us try to create a task with a different account:

Actions

Select a user: Agent ▾

Init

* name: Clean the wires

* amount (ETH): .01

Add task

transaction TEMP_1641664663953
Returned error: VM Exception while processing transaction: revert Only creator can call this function

Smart contract does not allow us to do so, modifiers allow only the creator to add tasks. Also user interface does not allow to submit unfilled forms:

Select a user: Creator ▾

Init

* name: Perform wire replacement

* amount (ETH):

'amount' is required

Add task

Each new task appears in the task section, contract information gets updated on the fly:

Tasks

- To do
Check electricity - 0.05 ETH
- To do
Clean the wires - 0.01 ETH
- To do
Check electricity schema - 1 ETH
- To do
Perform wire replacement - 2 ETH
- To do
Check remaining parts and finalize the process - 0.4 ETH

Contract Information

Balance: 0 ETH

State: Init

Combined task value: 3.46 ETH

Protection percent: 30%

Protection value: 1.038 ETH

30% is the protection percent that is set on initial contract deployment, which is case of this contract would be $3.46 \text{ ETH} * 30\% = 1.038 \text{ ETH}$

Let us start the deal

Init



* name:

* amount (ETH):

Add task



Start deal

Another feature that is implemented is event tracking. There are several events that a contract can emit, one of them is **DealStarted**. Notifications will show all events as well.

 transaction 



0x20a5abe5585b94135d00be33982d
32fd2a4febd3e6af333a6fecc3cd7933
41e2

Transaction is pending

 transaction 

0x20a5abe5585b94135d00be33982d
32fd2a4febd3e6af333a6fecc3cd7933
41e2

Transaction is successful! Block number: 12,
gas used: 45599

 Event: DealStarted 

No parameters

Event section displays all the events even after refresh:

Events

DealStarted

No parameters

Contract has changed its state to **ProtectionFromClient**, as we can see in contract info section:

Contract Information

Balance: 0 ETH

State: ProtectionFromClient

Actions section highlights (with the blue color) all the contract transactions that are available at that state. In this case, a single transaction **Send protection money** is available to the current state and the next state, which is **ProtectionFromAgent**. A small remark about **Send protection money** action - frontend automatically sets required protection amount as transaction value.

ProtectionFromClient, ProtectionFromAgent

Send protection money

PayForTask

Pay for task

Let us send the protection money at first from client's address

Select a user: Client ▾

Init

* name: Check remaining parts and finaliz

* amount (ETH): .4

Add task

Start deal

ProtectionFromClient, ProtectionFromAgent

Send protection money

And then from Agent's one. As a result, transactions were included into blocks and notifications were received. Also contract moved to the next state, as can be seen from the interface:

Select a user: Agent

Init

* name: Check remaining parts and finalize the process

* amount (ETH): .4

Add task

Start deal

ProtectionFromClient, ProtectionFromAgent

Send protection money

PayForTask

Pay for task

transaction

0x86574286681c1045ba2cf6a80c6411de3e5211eb1785f2318874d41b66053a41

Transaction is pending

transaction

0x86574286681c1045ba2cf6a80c6411de3e5211eb1785f2318874d41b66053a41




Transaction is successfull! Block number: 14, gas used: 34534

Let us check ganache for account balances:

ADDRESS	BALANCE
0x2Bce4befba07161f451624F947CE2f4c80ad36C5	99.94 ETH
ADDRESS	BALANCE
0x0f3cbb586128c7FdbF91E260143cB233d6C5eb15	98.96 ETH
ADDRESS	BALANCE
0xEA0E8d6De4f76259dcA739Ae8e8832e6aa635628	98.96 ETH

Indeed, $100 - 1.038$ (protection amount) = 98.962 ETH

That can be seen in accounts section as well with the more precise balance:

Actors	
Authorized consultant	
	0x2Bce4befba07161f451624F947CE2f4c80ad36C5 99.937968562 ETH
Agent	
	0x0f3cbb586128c7FdbF91E260143cB233d6C5eb15 98.961880772 ETH
Client	
	0xEA0E8d6De4f76259dcA739Ae8e8832e6aa635628 98.96188402 ETH

Another change to be observed is task section, 1st task got highlighted, indicating it is ready to be worked on:

Tasks

- 1 To do
Check electricity - 0.05 ETH
- 2 To do
Clean the wires - 0.01 ETH
- 3 To do
Check electricity schema - 1 ETH
- 4 To do
Perform wire replacement - 2 ETH
- 5 To do
Check remaining parts and finalize the process - 0.4 ETH

As a client, we first pay for the task, which is 0.05 ETH

PayForTask

Pay for task

Checking ganache indicated that client has indeed paid for the task - $98.96 - 0.05 = 98.91$ ETH:

ADDRESS	BALANCE
0x2Bce4befba07161f451624F947CE2f4c80ad36C5	99.94 ETH
ADDRESS	BALANCE
0x0f3cbb586128c7FdbF91E260143cB233d6C5eb15	98.96 ETH
ADDRESS	BALANCE
0xEA0E8d6De4f76259dcA739Ae8e8832e6aa635628	98.91 ETH

There is another valuable property of the contract that we can see in the interface - contract's balance:

Contract Information

Balance: 2.126 ETH

Let us verify the numbers:

Protection value * 2 + First task's value = $1.038 * 2 + .05 = 2.126$ ETH. Correct!

Once agent is done, he / she can send the task to revision:

TaskInProgress

Send task to revision

Cancel deal

Although, that is not the only possibility. Deal can be cancelled during this contract state, following the state diagram (that will be covered in scenario 2).

Now task evaluation process has begun:

Tasks

- 1 Evaluating
Check electricity - 0.05 ETH

From client's perspective there are 2 possible actions now:

TaskEvaluation

Accept task

* revisionMessage:

Decline task

Let us try to decline it with an appropriate message:

* revisionMessage: 2nd floor was skipped, please check it as well

Decline task

As a result, another event got emitted:

Events

DealStarted

No parameters

TaskDeclined

title: Check electricity, revisionMessage: 2nd floor was skipped, please check it as well

And again, task (contract's state) is set to **In progress**

Tasks

1

In Progress

Check electricity - 0.05 ETH

This time client accepted the task and event got emitted:

i

Event: TaskAccepted
title: Check electricity

X

Task panel:

Tasks

✓

Finished

Check electricity - 0.05 ETH

2

To do

Clean the wires - 0.01 ETH

What happened to balances?

ADDRESS	BALANCE
0x2Bce4befba07161f451624F947CE2f4c80ad36C5	99.94 ETH
ADDRESS	BALANCE
0x0f3cbb586128c7FdbF91E260143cB233d6C5eb15	99.01 ETH
ADDRESS	BALANCE
0xEA0E8d6De4f76259dcA739Ae8e8832e6aa635628	98.91 ETH

Agent received money for completing the task, $98.96 + .05 = 99.01$ ETH.

Let us fast forward to the moment just before the last task completion. Task panel displays the following task states:

Tasks

- ✓

Finished

Check electricity - 0.05 ETH
- ✓

Finished

Clean the wires - 0.01 ETH
- ✓

Finished

Check electricity schema - 1 ETH
- ✓

Finished

Perform wire replacement - 2 ETH
- 5

Evaluating

Check remaining parts and finalize the process - 0.4 ETH

And this is the balance:

ADDRESS	BALANCE
0x2Bce4befba07161f451624F947CE2f4c80ad36C5	99.94 ETH
ADDRESS	BALANCE
0x0f3cbb586128c7FdbF91E260143cB233d6C5eb15	102.02 ETH
ADDRESS	BALANCE
0xEA0E8d6De4f76259dcA739Ae8e8832e6aa635628	95.50 ETH

After the completion of the last task, there is an automatic protection money returnal. Let us calculate how much ETH each account will have:

For customer: $95.50 + 1.038$ (Protection money) = 96.538 ETH

For agent: $102.02 + 0.4$ (Last task value) + 1.038 (protection) = 103.458 ETH

Let us check that:

ADDRESS	BALANCE
0x2Bce4befba07161f451624F947CE2f4c80ad36C5	99.94 ETH
ADDRESS	BALANCE
0x0f3cbb586128c7FdbF91E260143cB233d6C5eb15	103.46 ETH
ADDRESS	BALANCE
0xEA0E8d6De4f76259dcA739Ae8e8832e6aa635628	96.54 ETH

Indeed correct! Let us verify balances on contract scale, not each task's scale.

From contract information we can get combined task value, which is:

Combined task value: 3.46 ETH

For agent: $100 + 3.46 = 103.46$ ETH

For client: $100 - 3.46 = 96.54$ ETH

Exactly what we were supposed to get.

Let us finish the contract by evaluating it

ReviewsAndRatings, ProtectionMoneyBack

* rating: ★ ★ ★ ★ ★

* review:

Amazing job

Write review

Deal is completed! Congratulations.

What we have now is:

Actors

Authorized consultant

0x2Bce4befba07161f451624F947CE2f4c80ad36C5

99.937968562 ETH

Agent

A

0x0f3cbb586128c7FdbF91E260143cB233d6C5eb15

103.45948276 ETH

Client

C

0xEA0E8d6De4f76259dcA739Ae8e8832e6aa635628

96.538723026 ETH

Contract Information

Balance: 0 ETH

State: Done

Combined task value: 3.46 ETH

Protection percent: 30%

Protection value: 1.038 ETH

Tasks

✓ Finished

Check electricity - 0.05 ETH

✓ Finished

Clean the wires - 0.01 ETH

✓ Finished

Check electricity schema - 1 ETH

✓ Finished

Perform wire replacement - 2 ETH

✓ Finished

Check remaining parts and finalize the process - 0.4 ETH

Interface with all the completed tasks, up-to-date balances, including balance of 0 for the address associated with the contract.

And all the events, where we can see the rating and the review:

Events

DealStarted

No parameters

TaskDeclined

title: Check electricity, revisionMessage: 2nd floor was skipped, please check it as well

TaskAccepted

title: Check electricity

TaskAccepted

title: Clean the wires

TaskAccepted

title: Check electricity schema

TaskAccepted

title: Perform wire replacement

TaskAccepted

title: Check remaining parts and finalize the process

RatingLeft

rating: 5, review: Amazing job

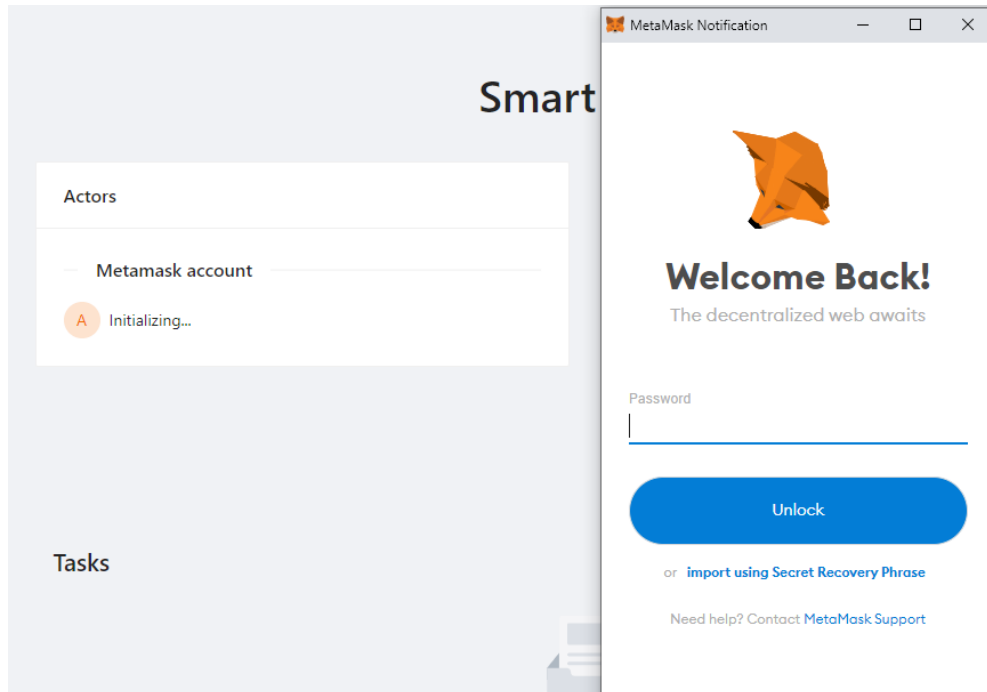
DealCompleted

No parameters

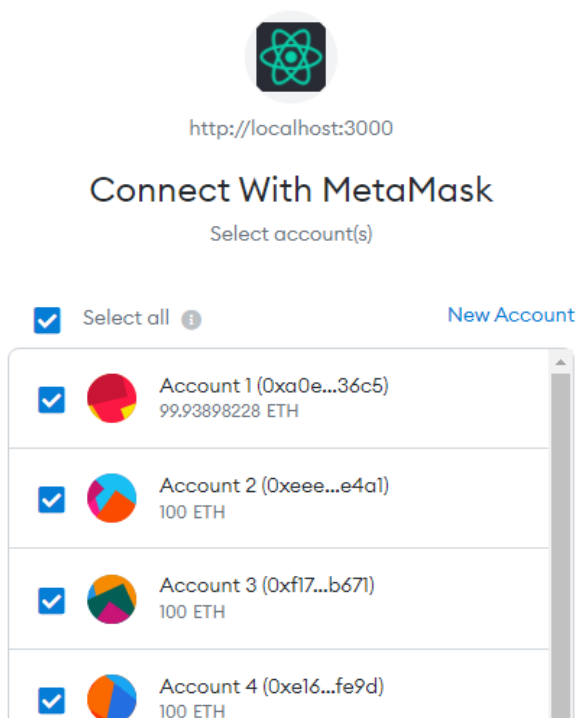
Interaction, scenario 2 - client cancels the deal (Metamask's provider)

For this scenario I will use Metamask to pick an address to send transactions from. Metamask will also show transaction confirmation pop-ups.

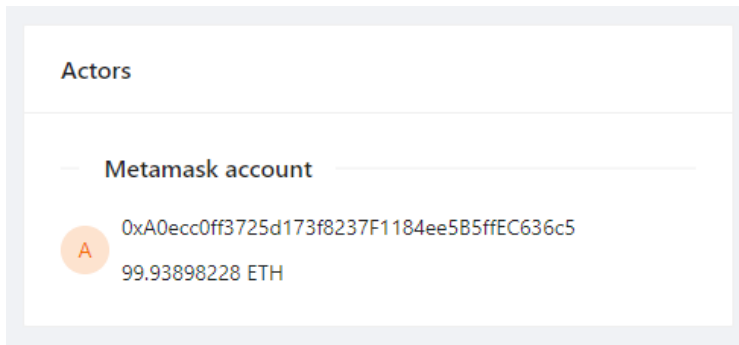
The first thing we see is metamask asking to log in, also Actors section displays Metamask account as pending.



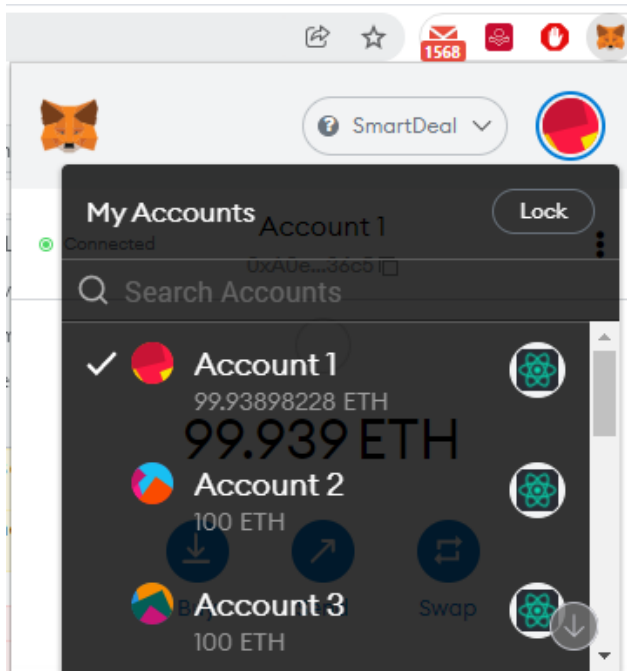
Use ganache's mnemonic to import the accounts, make sure you import all of them



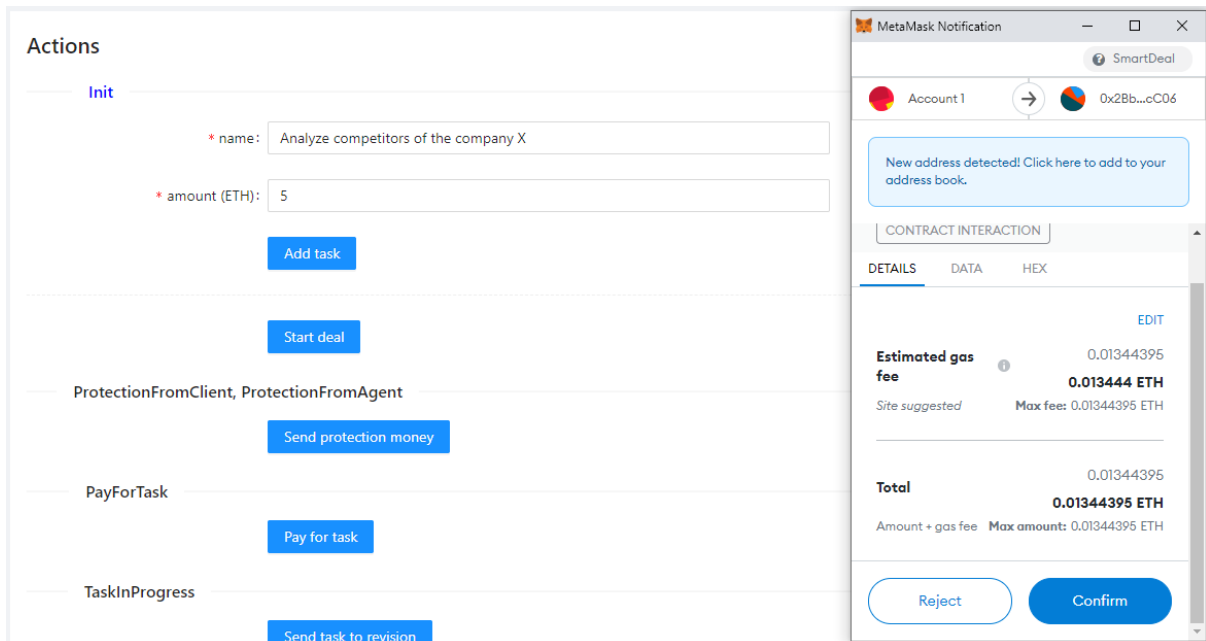
Now we are ready to go



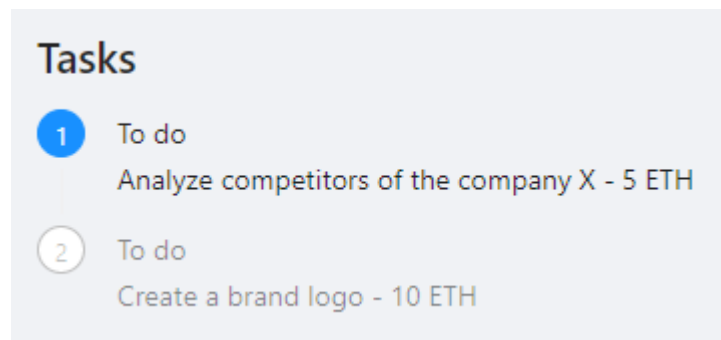
Let us create in a similar manner to the first scenario several tasks. As we are using Metamask now, accounts should be selected via extension.



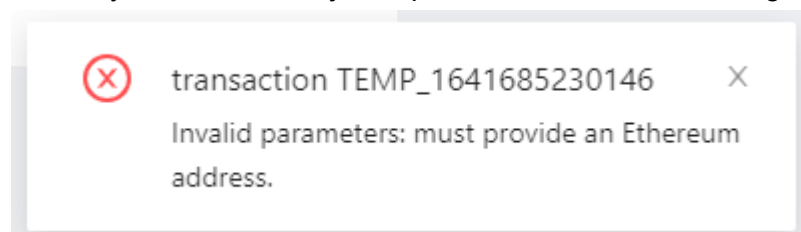
Add a task from the 1st account and confirm the transaction.



Once tasks are created, send protection money from both parties (and remember to change accounts)



Sometimes when you try to send transactions from user interface immediately after switching accounts, Metamask will display the following error, indicating that accounts haven't yet switched. So just repeat the transaction once again.



After paying for the first task, this is the contract's state

Contract Information	
Balance:	14 ETH
State:	TaskInProgress
Combined task value:	15 ETH
Protection percent:	30%
Protection value:	4.5 ETH

And these are all the balances:

ADDRESS	BALANCE
0xA0ecc0ff3725d173f8237F1184ee5B5ffEC636c5	99.94 ETH
ADDRESS	BALANCE
0xEe9B1C5E6820e9ED18F26926402044d33E1e4a1	95.50 ETH
ADDRESS	BALANCE
0xf17D11331cbE4Ff1eE2608607845a7AfCfDEB671	90.50 ETH

And now let us cancel the deal from the agent's account. What should happen is:

1. Value that got paid for the current in progress task should get returned back to the client.

2. As the agent has cancelled the deal, the client should get compensated for that.

So the agent's balance stays the same, while the client's balance increases by 14 ETH, which is the current contract's balance. That is $90.50 + 14 = 104.50$ ETH

Account 2

→

contract

http://localhost:3000

CANCEL DEAL

DETAILS

DATA

HEX

EDIT

Estimated gas fee

0.01344395

0.013444 ETH

Site suggested

Max fee: 0.01344395 ETH

Total

0.01344395

0.01344395 ETH

Amount + gas fee

Max amount: 0.01344395 ETH

Reject

Confirm

Confirm transaction. Another great feature that appears in task panel now is an indication of cancelled task:

Tasks

×

Cancelled

Analyze competitors of the company X - 5 ETH

2

To do

Create a brand logo - 10 ETH

Let us check the accounts:

ADDRESS	BALANCE
0xA0ecc0ff3725d173f8237F1184ee5B5ffEC636c5	99.94 ETH
ADDRESS	BALANCE
0xEe9B1C5E6820e9ED18F26926402044d33E1e4a1	95.50 ETH
ADDRESS	BALANCE
0xf17D11331cbE4Ff1eE2608607845a7AfCfDEB671	104.50 ETH

Indeed, balance aligns with our expectations. Now the client can write a review and close the deal. All events connected to the contract are accessible to everyone now

Events

DealStarted	DealCancelled
No parameters	initiator: 0xEe9B1C5E6820e9ED18F26926402044d33E1e4a1
RatingLeft	DealCompleted
rating: 5, review: good	No parameters

We were able to go through both scenarios and see how the smart contract was being executed and how user interface reflected the state of the contract.

Future improvements

There following enhancements can be done to “Smart deal” to make it more advanced tool for managing payments:

1. Introduce a time limit for task completion. If a task is not completed by the end of the agreed time period - deal gets cancelled and both money for the task and protection money of parties go to the client.
2. Add requirements for customer's balance before creating a smart deal, for example, a deal can not be started if a customer does not have the amount equivalent to protection money + money for the first task.
3. Introduce tips for greatly implemented tasks / a deal.
4. Introduce change management to “Smart deal”. Originally it was intended to create a separate “Smart deal” that helps solving changes to the project. Maybe in some cases it is more convenient to change deal parameters (shift deadlines, add / remove / adjust tasks) to fit requirements and keep everything in one place, rather than having several deals.
5. Pay protection money back to the client only after evaluation is done.