Jur - MVP

Brief

December 2018 by Jur Team

JUR AG, Grabenstrasse 25
Baar - Zug (Switzerland)
CHE-330.514.446
info@jur.io - www.jur.io

Abstract

In this document we are going to specify the MVP requirements. We will be going through some crucial flows of the platform and have a tour of a wireframe prototype that will guide you through the work to be done

Brief

Scope of the work is to build the JUR MVP. Jur is a blockchain based online dispute resolution system.

Jur is more than a mere dispute resolution system as the platform allows two party to actually create the contract on Jur and then if any problem arises they have an option of leveraging Jur's dispute resolution system.

In this first phase the platform will do the following:

- **Creation of a contract** between two parties (e.g. two individuals, two companies and so on). A contract will basically include details of the counterparties, the object of the contract with clear KPI and how to verify that it has been delivered, attachments, penalty fees in case of dispute and who has to put the money). In this step the contract gets saved to the blockchain so that in case of dispute it is possible to check that the contract's content (KPI, resolution proof, attachments) is the actual one that the two parties agreed on

- **Funds transfer to contract** upon payment of the required funds through Metamask by both the parties the contract gets activated

- **Contract management** in all the various stages (draft, edit, send for approval, ongoing, closed, disputed)

- **Contract closing system** in which one party proposes to the other to complete the contract successfully and the other accepts or rejects it (thus opening a dispute)

- **Friendly dispute** in case the two parties have some kind of disagreement on how funds should be distributed but they come to a solution. This friendly resolution has to be proposed by one of the party and accepted by the other. At this stage the first party can edit the proposal (e.g. the counterparty has given some inputs and the first party wants to amend the proposal accordingly). This allows the parties to talk to each other and amend the solution as needed

- **Open dispute** in case the two parties cannot reach a common solution then either of the two can open a dispute that will be judged by the community of oracles to determine who is right

- **Dispute management** which basically consists in allowing people to see the dispute, comment and vote

- **Email based notification system** which will be optional for any user but highly suggested to users' to take advantage of it (to avoid losing any crucial event happening on the platform)

Jur Smart Contracts Specification

The latest version of Jur's smart contracts are available at the following URL

https://github.com/jurteam/mvp-smart-contract

Note: as the smart contracts were developed much before the MVP tech doc you might find some naming differences but rest assured names will be coherent within the same document

Before moving into the flows it is helpful to see an overview of what the smart contract do and don't do as the scope of the present work will be to complete what is missing in the smart contracts and offer an interface towards them.

Anyone can develop an interface that will interact with Jur's smart contract that is why we need to be sure to provide an added value.

The below list is not completed, feel free to ask any doubts.

Smart contracts currently have implemented the following:

- They store the contract object and relative status
- They allow both the parties to sign the agreement
- They allow the counterparty to reject an agreement proposal
- They allow the parties to amend the contract details before its acceptance
- They allow any party to propose to close the agreement
- They allow a party to propose a friendly resolution by amending the initial distribution conditions
- They allow a party to agree or not to a friendly resolution
- They allow a party to dispute a contract in which she is a valid party
- They allow an oracle to vote on an open dispute on either of the party or on the reject case
- They automatically extend the dispute duration in case there is no clear (the vote is tied) or valid majority (more than 5% of the votes were placed in the last 30 minutes)
- They automatically distribute tokens to the parties according to the winning dispute resolution proposal voted by oracles
- They automatically distribute tokens staked on the losing sides to those whom voted for the majority

Smart contracts **do not** do the following:

- They don't handle file uploads (e.g attachments for the contracts, evidences for the dispute) those should be handled offchain
- They don't have a concept of user/party, a party is equal to a ETH wallet address
- They don't handle email notifications but they do support an event system that can be catched to trigger notifications offchain

Flows

Here you will find the main application flows for the important uses cases of the platform, we preferred an agile way of expliciting those but any detail can be asked.

Note: please download the SVG files as Google Drive won't render them properly

Statues and FSM

The following should represent the finite state machine of the MVP. This diagram should illustrate all the possible statuses through which a contract will go through. This is an input and should be double checked before developing.

Diagram: https://drive.google.com/open?id=1OBewIY_bLImIxR9XPA5aF-8FAYjfde_n

Contract Creation

This flow is quite simple either Alice or Bob creates a contract. They are allowed to save it as a draft (for the scope of the MVP an autosave feature is not required) and/or send it to the counterparty. Once they send it to the counterparty the contract gets stored on the blockchain.

Flowchart: https://drive.google.com/open?id=1sDYONaejGbZr7OMTvAbZrFlUxo2PODWK

Contract Acceptance

This flow is quite complex as finally we are onchain. The counterparty can accept/reject the proposal. If the counterparty accepts we move on in the flow. If the counterparty rejects the contract will be closed.

According to the smart legal contract specification either or both Alice and Bob should pay their funding fee. Once that is settled according to their requirements the contract is considered to be ongoing.

Flowchart: https://drive.google.com/open?id=1wJwkaxC9QMusrnaHbiWIDTggyYAOmzYE

Contract Closure

In this flow the contract is either expired or Alice or Bob wants to close it prematurely. The closing event means that the proposing party signs off through Metamask the closure.

If the counterparty signs as well then the contract is considered to be closed.

The counterparty has the option to reject which will lead to the dispute flow.

Flowchart: https://drive.google.com/open?id=1v6G93hIxcYXRuJXUD2KdUbQwkF8kJ6Ah

Dispute

We reach the dispute flow in two ways: either the counterparty has rejected the party's closing proposal or at any point one of the two party felt that the contract could not close as per the initial contract's conditions (e.g. Bob has delivered less work, exchange rates changed drastically for an OTC transaction, etc.)

This is by far the most complex flow of the whole platform.

There are two options for a dispute it can be either *friendly* (closed to the parties) or *openly* (visible to all the oracles).

In the first case a party can propose to the other to close the contract. If the other party agrees the contract is withdrawn successfully.

In case the other party rejects the proposal, she has the option to open a dispute. While opening a dispute the party has to pay a fee of 1% and alongiside set her conditions of distribution of the tokens inside the contract (the one paid by the parties when the agreement got signed by both of them).

The counterparty has 24 hours to reply with her dispute resolution proposal (basically how to redistribute the tokens given the disagreement). If the counterparty fails in sending her proposal we assume that the proposal will be the maximum to her favor.

Now the dispute is considered to be opened and any oracle can vote either for *Alice*, *Bob* or *Reject*.

Flowchart: https://drive.google.com/open?id=12ur9H4uanrus9hTEJKFIqpgxGS56x6IX

Voting

In the voting flow the actor can be either Alice, Bob or an oracle. The overall logic happens directly in the smart contracts so from the offchain perspective there is nothing much to worry.

The need is just to call the appropriate methods to register a vote on that specific dispute.

All the extension logic will be managed by the smart contract but of course we should be listening to those events to refresh our UI (e.g. the dispute has reached the end but there is no clear majority so it gets extended by 30m)

https://drive.google.com/open?id=15nd60nkXE0hJIshJIvj0ZWclYUmiMjlH

Stack

The overall technological stack will be the following:

- *NodeJS* / *ExpressJS* as backend
- *ReactJS* / *Redux* as frontend
- *Postgres* / *Sequelize* as DB stack
- *S3* for file uploads (*Digital Ocean Spaces*)

Some libraries that we suggest to use:

- *Passport* for authentication
- *Helmet* as security middleware
- *Webpack* for build pipeline
- *Dotenv* for environment variables management
- *Swagger* for documentation

As per the code quality these are our requirements:

- *Git* for version control, we follow these conventions
  - *Develop* branch deploys in the dev server
  - *Beta* branch deploys in the staging server
  - *Master* branch deploys in production
  - For commit messages we tend to follow this format ->
    - *Added: <message>*
    - *Fixed*: *<message>*
    - *Updated: <message>*
    - *Removed*: *<message>*
    - If multiple things are getting done in a single commit, *Added: 1) .... 2) ...*
    - If multiple actions are getting done in a single commit, *Added: <message> Updated: <message>*

- *Codacy* for code analysis, A grade certificate
- *EsLint* as a linter
- *Prettier* for code formatting
- Unit test and integration tests, we suggest *Mocha/Chai* but open for inputs a coverage of 80% will be considered good

No preferences over the deployment choices though our CI/CD pipeline will be working with *CircleCI*

As per the browser support we would like to support properly all browsers and respective versions supported by Metamask

Software Design

From a software perspective we would like the code to be structured using a multi-layered approach which will consist in:

- **Models**: which will contain our domain data
- **Controllers**: which will receive requests from the client, analyze it, if valid pass it to the services layer and return a response to the client
- **Services**: which will have all our business logic, this layer will talk directly to models and with the controllers
- **Helpers**: any utility function that is used across the layers
- **Middleware**: any utility component which will be used for all requests/response lifecycles

We also welcome separation of concerns so we suggest to split the business logic in separate apps accordingly (e.g. we can separate the uploader from the contract apps or the dispute one as it used across)

Architecture

As per the architecture we generally leverage a configuration of *Nginx* as a reverse proxy for the API exposed by *Node* but we are open for inputs.

Ideally we would like to separate the DB and the Application server and make sure that everything is replicated properly.

We do not foresee an immediate need for a load balancer.

Backup routines will be appreciated as the scope of the project is important to users.

Prototype

The following prototype was made to showcase the product to investors and to gather feedback from our community. The final product should be really close to this prototype so it should be easy to evaluate the complexity of the frontend as final graphics will be almost this ones. We have some missing view but which will be basically gathering already existing elements and should not add much work on the development side

https://invis.io/HCPK7A1GQEN

Please do enter the following password to access the Invision prototype:

207.154.202.13

Glossary

- **Contract's value**: this indicates the total funds transferred by Party A and Party B inside the contract on the blockchain through the Metamask interface. For example if Alice and Bob have smart legal agreement in which Alice has to put 2000 JUR tokens and Bob 500 as a collateral for his services then the total contract value is 2500.
- **Counterparty**: by counterparty we mean the opposite side to the creator of the contract (named party). Both of them will be usually represented by a valid ETH address
- **Dispute**: a dispute happens when one of the party does not agree anymore with the other on the validity of the initial stipulated contract and want to resolve it either in a friendly way or by having an open dispute open to oracles to vote
- **Dispute opening fee**: to open a dispute and involve the community of oracles the party that wishes to open the dispute has to pay a fee of 1% to incentivise the oracles to express their opinion quickly
- **Duration**: in case of a contract the duration states the time after which the contract will be considered expired and the parties are urged to either close it successfully or to open a dispute. In case of a dispute the duration indicates for how long the dispute will last and for how long oracles will be able to vote. As a reminder a dispute duration might get extended as the result of a non clear/valid majority
- **Evidence**: an evidence is any attachment that either of the parties uploads according to their dispute resolution proposal
- **Friendly Resolution**: a friendly resolution is a dispute in which the parties are able to reach an agreement on how to distribute the funds in the contract without having to reach out to the oracles community
- **Hub**: an hub consists in a group of oracles that follow a set of legal principles while judging an open dispute. For example we might imagine one day that there will be a kind of International Freelancer

Association Hub that will have a common legal framework throughout the world and will be judging any smart legal contract which has listed the hub in case of dispute.

- **KPI**: is a field inside a smart legal contract that states clearly (in a textual format) what is the content of the contract between two parties. This allows an oracle to quickly have a sense of the object of the contract before going deep into the issue. KPI should clearly state in an objective way what each party has to do to honorate the contract
- **Open Dispute**: this type of dispute happens when the parties are not able to reach an agreement while closing the contract. In this case both of them will propose their solution to the contract and a community of oracles will judge their proposal and declare who is right
- **Oracle**: anyone who possesses at least 0.1 JUR token. This allows her to express her opinion on any open dispute on the Jur platform. Usually an oracle is represented by an ETH wallet address
- **Reject Vote**: a reject vote is a vote that the oracle places on neither of the two parties. This means that basically the oracle believes that the object of the contract is either illegal or was malformed in the initial stage so it is difficult now to express a valid opinion
- **Resolution Proof**: is a field inside a smart legal contract that states clearly (in a textual format) how an oracle can check the deliverables of the contract. This should be a set of instructions in how anyone can check that the the deliverables were completed according to the KPI
- **Reward**: by reward we mean the tokens that will get distributed at the end of a dispute. This reward will be including the initial 1% fee paid by the opening party and all the tokens staked on the losing parties
- **Smart Legal Contract**: it's a contract created on the Jur platform that allows to party to store the details of their agreement and access the Jur's online dispute resolution system