CATAPULT Digital

Blockchain for the creative industry

BACKGROUND

Blockchain was the buzz technology of the last year. Everyone started talking about it, companies doubled their value by just mentioning they use it, and initial coin offerings (ICOs) popped up everywhere. At the same time, it was increasingly clear that not many blockchain-based products were making it to market or were proving to be commercially viable.

To help companies understand blockchain and separate the value from the hype, Digital Catapult has grown an internal team to design and build full-stack blockchain applications. As part of our creative industry programme, Digital Catapult has collaborated with the UK Games Fund and developed a user-centric product, which allows early-stage, ad-hoc teams of indie game developers to agree on ownership for a specific game and to record their decisions on a public blockchain.

The alpha version allows creative teams to allocate different types of shares for an ad-hoc project, add new team members, and vote on proposals using a transparent and tamper-proof process. It also allows the teams to manage disputes and involve agreed arbitrators to resolve future disagreements between the team members.

Through this journey we have developed modules and reference designs, learning many valuable lessons that we would like to share with the community. We believe that this project demonstrates many of the benefits of blockchain and also exposes many of the challenges that any company who wants to work in this space will have to face, such as deciding when to use blockchain and when to use traditional databases, which blockchain flavour to choose for a specific business use case, and so on.

Blockchain is the technology behind Bitcoin and other cryptocurrencies. It was designed initially to prevent double spending without a central point of authority, which was one of the biggest challenges in developing a decentralised currency system. The real hype around blockchain started when Ethereum was launched in 2015 with the purpose of allowing developers to build decentralised applications (dApps). We assume that the reader has a basic knowledge of blockchain and understands the different concepts behind it. If not, a list of important blockchain white papers and articles can be found here 67 blockchain articles and whitepapers that shaped crypto into what it is today.

At the beginning of the project we agreed upon these key guidelines:

- Use a public, permissionless blockchain network
- Use popular platforms and tools
- Use a production network (not a test one)
- User experience (UX) is important. We wanted to build a product for normal users, not just developers
- Properly engineered solution, cloud based (AWS), security in mind and fully tested

TYPICAL USE CASE

Imagine a team of game developers who have decided to design and build a game. Sarah and Matthew are software developers, Ben and Ran are animation and graphic designers. The team members believe they have a great idea for a game, but are not yet ready to set up a company together. Also, some of the team members might work on other games with other teams, or work elsewhere part-time.

As the team is confident their game is going to be a huge success, they want to agree on percentage ownership allocation beforehand to avoid future arguments and disputes. They want to publicly record their agreement and to allow other contributors or investors to join in their effort as it develops. They also agree to nominate Paul and Mark from UK Games Fund to help them resolve disputes in the future.

They decide on the following share allocation:

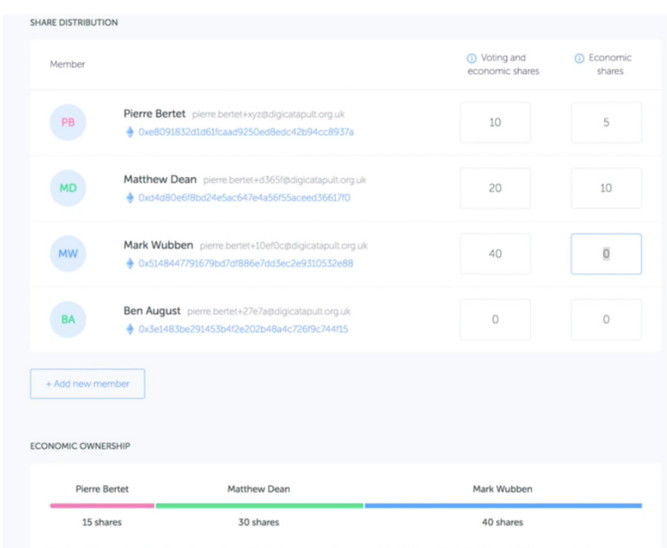
- Sarah: 100 shares
- Matthew: 100 shares
- Ran: 100 shares
- Ben: 50 shares
- UK Games Fund: Agreed arbitrator

At some point the team realise that they need another software developer to work on an Android version of the game. They invite Callum to the team and propose to give him 50 shares in return for this task. One member of the team creates a proposal and they can all vote on it. If more than 50% of the voting shares approve this proposal, Callum will receive his 50 shares. The team can decide to give Callum 25 shares before the job starts, and 25 shares upon completion of his task.

Similarly, investors can get shares in return for investing in the team and the game.

If the team votes on a proposal that is against the terms of the contracts, illegal or unfair, for example if the team decides to cancel Ben's shares, Ben can ask Paul and Mark from UK Games Fund to start a dispute resolution process which can enforce another proposal that gives Ben back his shares.

The full use case is explained in detail in the following white paper:



WHY BLOCKCHAIN ?

Three years ago, if the above use case had been described to professional software developers they would have probably suggested developing a product and implementing it using a three-tiered architecture: a database that holds the share allocations and user details, a backend which handles application-specific logic, API requests, authentication and authorisation, and a frontend that implements the user interface.

So why blockchain? Why do we need to add a component this complex to our system, run the same code on thousands of machines around the world, pay money for transactions, and wait minutes for them to be approved? What is the advantage that blockchain provides over existing products? The answer is that it's all about fraud risks, transparency and trust.

Existing online products which manage ownership of assets and rights all present multiple risks:

1. Cyber security risks: A hacker, internal or external, might get access to the database, modify some of the records and their backups leaving no evidence of the previous state and no way to recover from it.
2. User transparency: You, as a user, only see what the user interface presents to you. You have no way to know what is actually stored in the database.
3. Audit log and history transparency: Traditional databases only store and manage the current state. Systems might keep internal logs, which hold records of each transaction for many years, but these logs are usually just a dump of cryptic long text lines used for troubleshooting and are not human readable.
4. Hosting: The company who hosts the product might go bankrupt, they might delete your records or not care about them.
5. Change management: The product might change and the agreed upon processes when setting up the share structure are now different. Perhaps voting rules have changed or maybe the dispute resolution process is different. You have no control over the product and no way to prevent a company from changing the rules of the game.
6. Signing ceremony: There is no way for users to prove which actions and operations they approved as their signature is not attached to every transaction.

How much we care about these risks depends on how valuable the assets that the product manages are; essentially 'what is at stake'? As the value increases, users want to take fewer risks and have more guarantees that the product will protect them.

So why blockchain? When using blockchain as a data layer there is no way to erase the history. Once a transaction is approved and mined it is in the blockchain forever; everyone can see it, verify its content and its origin. Transparency, traceability, immutability and security are inherent features of any blockchain.

How do traditional companies solve these issues today? They spend a lot of money buying cybersecurity tools and products, they develop internal processes and replicate their data to make sure the service level agreement (SLA) is kept to.

There is a misconception that blockchain is a trustless system. It is said that you, as a user, don't need to trust anyone; neither your bank, nor your lawyer, and that somehow blockchain will protect you against all of these third parties. What actually happens is that you are asked to trust different third parties; the company who develops the product that you are using (e.g. a wallet provider), Ethereum/Bitcoin developers and finally mining pools.

So, it all comes down to trade-offs and risk management, just like any other engineering decision. When we build products that manage ownership of assets and rights we can use blockchain in order to get some of these benefits 'out of the box', or use traditional solutions and wrap them with the right tools and processes to get the same end result.

WHY A PUBLIC PERMISSIONLESS BLOCKCHAIN?

The difference between public networks and private ones is mainly related to whom is allowed to add a node to the network and participate in the consensus protocol. Anyone can run dApps over public networks and get access to the services that the network provides - such as mining, submitting transactions, and deploying contracts. Anyone can also be a miner on these network and maybe, if they invest enough resources, turn it into a profitable business. However, to setup a private network, you would normally gather together several companies or organisations in the same industry to create a consortium who share the same problem and interact with each other and then agree upon access permissions to the network. If a single company ends up running the network, most of the benefits that blockchain technology provides are removed because there is a "single point of failure" again. Public networks have their disadvantages as well. You will have to pay for each transaction and experience the results of events occuring on a network that you don't control, which might affect your operations.

More information on the differences between private and public blockchains is available in this article: https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/

For our project we chose to use a public permissionless network because it was easier and faster. Setting up a private network requires long term commitment from a number of independent companies/organisations in the games industry, and although this is probably the way forward, we wanted to prove the use case first.

WHY ETHEREUM AND WHY SMART CONTRACTS?

There are two leading public blockchain platforms in common use: Bitcoin and Ethereum (others are rapidly catching up, and we continue to watch this space.) We chose to use Ethereum because it's not just a blockchain platform, it also allows you to run arbitrary code (smart contracts).

There are two main advantages to running smart contracts on a blockchain as opposed to running the same code off-chain:

1. Smart contracts can safely hold cryptocurrency and make decisions as to who to transfer it to and when. For example, a user who buys a game can transfer Ether (Ethereum coin) to the smart contract and the smart contract can decide to split the amount of Ether between the team members based upon the share allocation in its state. Additionally, smart contracts can interact with other smart contracts, and activate methods that can transfer cryptocurrency or information to other addresses.
2. The smart contract code, just like any other data on the blockchain, is persistent and cannot be modified. Once the smart contract is deployed to the blockchain, it is final, there is no way to change it and any functionality that it includes is fixed and permanent. This means that when you agree to the service rules there is no way for anyone to manipulate them in the future. There are some schemes and design patterns which allow versioning of smart contracts, like proxy contracts or contract delegation, but they are not as simple as a traditional code upgrade.

Both of these advantages are balanced by concomitant risks. Firstly, what happens if we deployed our contract and now realise there is a bug in it or that it is missing functionality? More importantly, the code for the smart contract is publicly available and it could be just a matter of time until somebody finds a security vulnerability in it and extracts all the cryptocurrency from it. Trust No One: Ethereum Smart Contract Security Is Advancing

For these reasons we would recommend avoiding smart contracts unless they are really necessary. For many use cases, running code off-chain that relies on data on-chain is sufficient. This does centralise the governance of the system into the hands of the data processor, but also lowers the risks of smart contract failure during these early days of blockchain technology.

WHY A PRODUCTION NETWORK?

Ethereum exists as one production network and several test networks. Test networks are similar to the production network in almost every respect, but have fewer miners, less activity and much lower difficulty, so that it is purposely very easy to get Ether for testing purposes. As we are building a product mainly for demonstration purposes it makes more sense to deploy it in the test network and save money.

However, we realised that running smart contracts in the production network would expose us to many more interesting and surprising behaviours. Estimating the gas price (the cost of deploying or executing a smart contract) is not as trivial as it seems. There are many network effects like ICO sales and hard forks which one must be aware of. We wanted to experience these behaviours and learn from them. Another reason why we preferred to deploy our smart contract to the production network is that the Ethereum test network is unstable. There are many hackers who try to break it and test weird things on it with the goal of improving Ethereum overall. The goal of this project was to support real fledgling companies and we needed to use the real network to support this functionality.

BALANCE BETWEEN TRANSPARENCY AND PRIVACY

One of the key design decisions that we agreed upon at the beginning of the project was about which data to store on-chain and which data to store off-chain. Data we store on the blockchain is publicly available, transparent and immutable, whereas data that we store off the blockchain is private and confidential but is not immutable, so careful consideration

should be taken when making this decision. In light of GDPR it is critically important for any teams using public blockchains to consider ways to avoid storing Personally Identifiable Information (PII) on-chain.

We decided to store only the game name, game description and share allocations on the chain, whereas emails and names of the team members (the PII) were kept off the chain.

This means that anyone can query the blockchain and extract the ownership structure for every game that we submit. On the other hand, there is no easy way to know who actually holds the shares as there is no public directory which links blockchain addresses to emails or names. One could argue that the addresses to which a share allocation belongs may be considered linkable to PII, but the jury is still out on this point.

Off-chain data is stored in a standard postgreSQL database hosted securely in our private AWS account where only authorised services can access it.

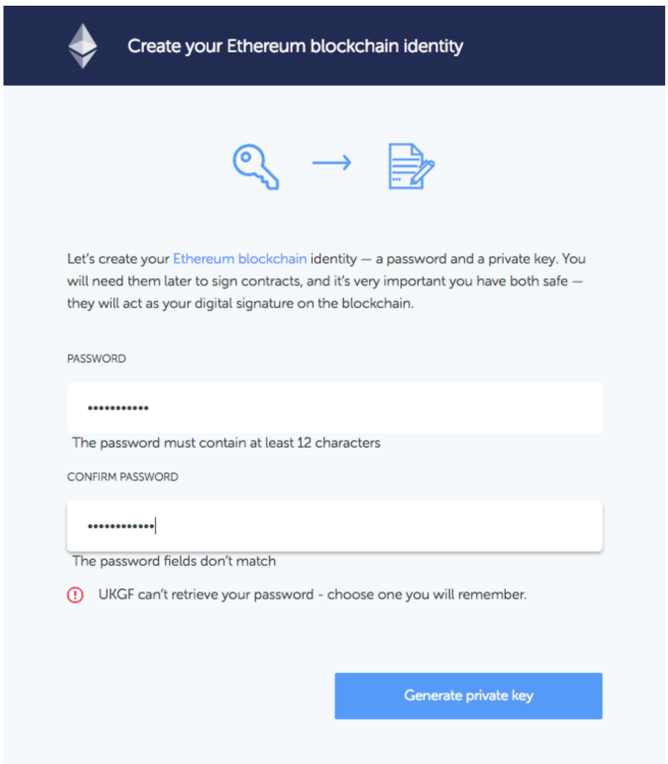
PRIVATE KEYS

Whenever users submit transactions to the blockchain, they need to sign them using their private keys (signing ceremony). Without private keys there is no way for a user to vote, transfer shares or create proposals. The signature is also used to confirm that the transaction has come from the user and not from anyone else. Private keys are very hard to remember as they are necessarily long, meaningless sequences of numbers and characters. For this reason they must be stored somewhere private and protected, usually using a password or some sort of hardware device.

Unfortunately, when users lose their private keys, unlike existing online products, there is no way to recover them or reset them. Storing a backup of the private key on an off-chain server is a serious cybersecurity risk, so the only alternative in these situations is to create a new private key for the user. As the blockchain address is derived from the private key, somehow the shares of the user under the previous address must be transferred to the new address.

Losing private keys is an unavoidable scenario, therefore dApps must include a secure and trusted mechanism to protect the assets and rights of users when it happens.

In our implementation, transferring shares between an old key and a new key is handled just like any proposal. The team, once informed of this situation, can vote to move all of the shares from the old account to the new account. In cases where the team is not cooperative, or when the user holds the majority of the voting shares, an arbitrator can enforce a proposal to move the shares.



USER EXPERIENCE AND PERCEPTION

How users interact with the blockchain and what they perceive of the process was a key question for us to investigate and understand throughout the project. We conducted several user testing sessions and realised that there is a big gap between the perceived benefits of blockchain and what users actually understand is happening behind the scenes and the risks they take.

Generally speaking, users trust applications far too much: when asked to load a private key or to sign a transaction they will cooperate without verifying what it means. We believe that it is dependent upon the application developer to build a process which is on one hand secure and transparent, and is on the other hand simple to use. Users of an ideal blockchain product will not be aware of the technical details of what happens behind the scenes or even what blockchain means but will still appreciate that the product they use is more trusted and secure.

We discovered that the experience and perceptions of a user changes when using a blockchain app, which exposed important gaps.

In terms of experience:

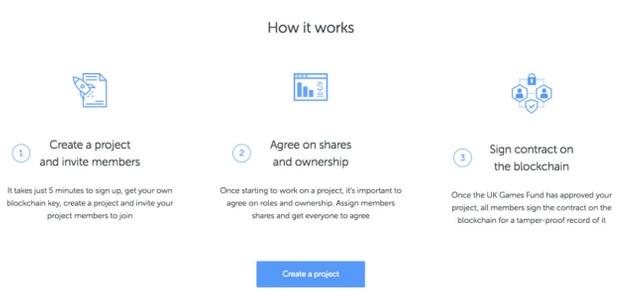
1. To interact with blockchain, vote and sign, transaction users need Ether in their account. Buying Ether is a complex and long process even for technically capable users.
2. Users need to remember two passwords and to know when to use the correct one: the first to sign in to the service and the second to unlock the private key; one can be reset and one cannot.
3. Private keys, unlike passwords, are long and meaningless numeric sequences. There is no way for users to remember them correctly,

so they must be stored securely for many years. We found that most users just store them as files on their disk/laptop without creating backups and forget about them.
4. 'Ethereum address' is another confusing term for users. They are similar to bank account addresses, but are visible to everyone.
5. Users don't necessarily understand that operations on the blockchain are not immediate, they can take minutes to be completed and some operations might fail.

Perception:
1. Some of the data that users provide will be publicly available forever without any means to delete it.
2. Blockchain has a bad reputation for hackers and illegal activities. We don't trust it.
3. Cryptocurrencies are unstable. Their value changes dramatically every day.

To mitigate some of these gaps, we decided on the following UX strategy:

1. Users don't need to buy Ether. The product automatically transfers a small amount of Ether to their account so that they can interact with the blockchain and submit transactions. As we are transferring only a small amount of Ether to each account, the chances that users would steal the Ether and use it for another purpose are relatively low.
2. As users are not aware of the Ether being used behind the scenes, they don't need to care about or be aware of Ether price fluctuations.
3. Whenever the user interacts with the blockchain, we substantially change the design of the web page to highlight the importance and criticality of the user's actions, thus creating the perception of a signing ceremony experience.
4. The private key is stored securely in the browser so there is no need to upload it for every transaction. This was a compromise made between the risk of key loss versus security.



SMART CONTRACT DESIGN

It may seem at first glance that designing smart contracts is similar to designing any other object oriented software. You have state and code and functions that modify it. Solidity, which is the most common programming language for writing smart contracts on Ethereum, is similar to Javascript or C in many ways, so the general assumption is that any experienced software developer can produce a smart contract fairly quickly. However, after writing a few lines of code in Solidity developers will start to realise that this programming language is unlike any other they have worked with before.

Language Limitations: Solidity is a domain specific language whose purpose is to provide a somewhat JavaScript-like language which abstracts the low-level details of the Ethereum Virtual Machine (EVM) from the user. Unfortunately, there is a strong debate within the community as to whether such a language is really what smart contracts should be written in. Certainly the language as it stands has some pretty big gotchas which have been written about at length, for example:

- How to secure your smart contracts six solidity vulnerabilities and how to avoid them part one
- How to secure your smart contracts six solidity vulnerabilities and how to avoid them part two
- Solidity gotchas

From a developer's point of view this means you have to be really careful and invest the time and effort to test and audit your contract code for security vulnerabilities.

Contract size: Deploying a contract is no different from submitting any other transaction to the Ethereum blockchain. This means that contracts must fit within a block and that the contract size should be executable within the block gas limit. With the current block gas limit of 8 million 'gas' this means the maximum contract size is about 29KB. When deploying contracts larger than this, developers should split the contracts or use libraries. This in turn comes with its own risks, as highlighted by the parity multisig wallet bug from November 2017 (http://paritytech.io/security-alert-2/).

Gas estimation: Ethereum is able to execute whatever code is stored in a target smart contract. Ethereum code is arbitrary and Turing complete meaning that ahead of time there is no way to know how much computation will be done by the smart contract or even if it will finish (https://en.wikipedia.org/wiki/Halting_problem). To make sure that the execution of a smart contract doesn't grind the whole network to a halt, each transaction is charged based upon the amount of computation consumed. This is done in two steps; the contract caller pays an amount based on their estimate of the maximum cost of execution (the 'gas limit'). The code is then run. If the code starts to cost more than the gas provided then execution is halted, any change to the state is rolled back, but the caller is still charged. If the execution of the smart contract code uses less than the gas provided, the caller is refunded the difference.

This overcomes the impossible task of having to check if a transaction will finish by making the user place limits on transaction execution. Now though, the user has to decide on these limits. A lot of the time this is easy; do a dummy run of the execution and see how much it cost. But what happens when the cost is dependant upon other transactions or runtime data such as block hashes? This might sound abstract but in a voting system an early vote is potentially very different in cost to the late vote that accepts or rejects a proposal. When choosing a gas limit, the developer needs to know the context of the call and estimate the maximum possible cost of the transaction. For this project we simulated

worst case costings for each transaction type and used those as limits. The process for this will, however, be application-dependant and may inform contract design.

Gas estimation is only half of the problem when trying to price a transaction. The gas limit sets how much computation may be performed, but this must also be priced in Ether (the base currency of the Ethereum blockchain). In order to introduce competition between transactions the user also sets a price in Ether that they are willing to pay for their gas. This means the final transaction cost is given by:

Transaction_cost = gas_used × gas_price(in Ether)

When a miner is picking transactions to include in a block (including transactions that call smart contracts) they will start with those that have the highest gas price as this will be part of their reward. A developer therefore needs to have an idea of gas market conditions in order to set a price for a given transaction. One option is to leave this to the user, but this exposes an end user to strange concepts they are unlikely to understand. Another is to dynamically set the gas price using a service like ETH Gas Station https://ethgasstation.info/ but this is a centralised service and hence prone to corruption or 'rigging'. The best option is to use deployed infrastructure that does this for your app, and options for this have started to recently spring up, e.g. https://github.com/ethgasstation/gasstation-express-oracle. All things considered, there is still no best practice way to do this. As a developer you must choose which approach to take, considering the likely complexity of your final code and the user experience you are seeking to provide.

Agile / Waterfall: there is no MVP (minimum viable product) with smart contracts on Ethereum. Once a smart contract has been deployed, it's very hard to modify it. This means that software developers need to be really careful when using standard agile processes such as continuous delivery. A Waterfall approach somehow feels better when developing smart contracts: you want to understand all the requirements and scenarios and document them, review them several times, put everybody in the same room and think about all the edge cases, what needs to be configurable, who should have access to what, what data is private, what is transparent and what should be hashed. You then need to write a test document, which tests every feature and covers every line of code. Formal verification might also be important if your contract holds or deals with any amount of money. Finally, before deploying to production you should re-review the process, review the code again, and get everybody in the team to sign off on it.

Smart contract upgrades are not impossible though. In our design, we implemented a way to upgrade our code and transfer the state to a new contract. Contract upgrades must be approved by all team members who have voting shares and some of them might refuse to do so for many reasons. Also, this defeats the principle that 'code is law' and that code should be immutable, and opens a backdoor for malicious hackers who can leverage it to hack the contract and extract the money it holds.

Contract Roles vs. Users: in some situations it is desirable to have different roles within a contract and assign users to these roles. For example, in the UK Games Fund use case there are two official arbitrators: Paul and Mark. Both of them can manage, dispute and enforce proposals. In the future there might be a need to add more arbitrators or change the current ones. The implementation we chose was to create a proxy contract that we called a 'gold' contract. The gold contract has several responsibilities:

- Bank: It holds a small amount of Ether and transfers an initial amount to the team members upon creating the game contract.
- Factory: It creates the actual game contracts.
- Role Proxy: It holds the list of arbitrator users and proxies their requests to the game contracts. This way, the game contracts only see one arbitrator (the gold contract) and are not aware of the actual arbitrator who initiated a transaction.

Multiple simultaneous proposals: An interesting edge case is what happens if there are several pending proposals. Some proposals may, once accepted, change the number of voting shares. We had many discussions on what to do in these situations. Should we use the updated number of voting shares across the other pending proposals, or should we use the number of voting shares a user had based on the time that the pending proposal was created. Finally, we decided that if any proposal changes the voting shares, all pending proposals would be reset and members would need to vote on them again.

To conclude, smart contracts are an interesting and compelling concept but require a substantial amount of technical investment and support. When deciding to incorporate smart contracts in any system, it's advised to carefully weigh the advantages and reasons behind it.

ENGINEERING A BLOCKCHAIN PRODUCT

25 years ago, before we had the Web and online products, software developers had the luxury of deploying enterprise software in a fixed and controlled environment. They could define the devices customers used their software on and set up the network configuration that supported it. As the internet evolved and online products emerged, software developers had to adapt to new scenarios where they could not predict the devices their customers would use their software on and how their network connection was going to behave.

Looking at this from a three-tiered architecture perspective (Presentation-Application-Data) developers could not control or define the link between the presentation layer and the application layer. Imagine what happens when users who use an online product lose their internet connection or change their internet access method. Messages between the frontend and the backend disappear, they arrive in a different order or are duplicated. Experienced frontend and backend developers learned to handle these edge cases and wrapped their software from both sides to accommodate them.

On the other side of the system, backend developers could rely, most of the time, on expected behaviour between the application layer and the data layer. Databases were tuned and sized to reach the desired performance, and integration was fairly simple and reliable.

Blockchain changes all of this. When using blockchain as the data layer, backend developers now need to worry about unexpected behaviour on this link as well. When submitting transactions to the blockchain there is no guarantee that they will be accepted, for example, in the case of setting too low a gas price. Some transactions might be stuck in limbo for a few days and some might be accepted, but potentially after a few blocks they will have been rejected. There are two different states: what you write to the blockchain, and what you read from it. This inconsistency can sometimes last for a few minutes and needs to be managed as it can be somewhat confusing for the user.

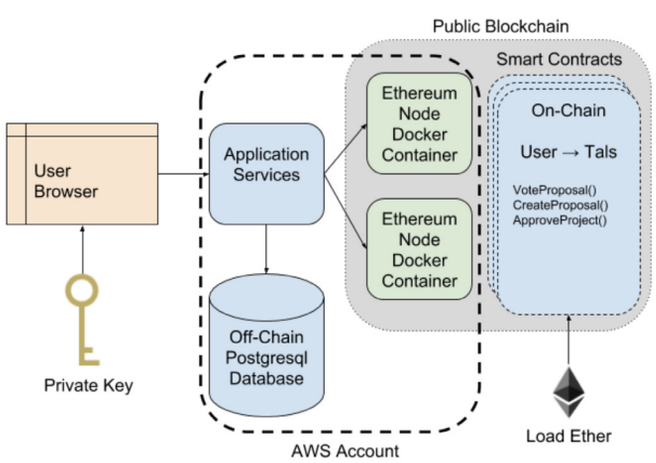
In our implementation we have developed a Transactor service, which handles all of these edge cases. It submits transactions to the blockchain node, tracks their progress, resubmits them if needed and filters rejected ones. In this way the unexpected behaviour is masked from the application.

Furthermore, we have developed a contract watcher service, which monitors all the events from the gold contract and the game contract, parses them and pushes them to a message queue in AWS.

OPEN SOURCE AND COMMUNITY SUPPORT

Both of these services and additional packages have been open sourced and are available on our Digital Catapult Github account: Digital Catapult GitHub.

Transaction Watcher: ethereum-transaction-watcher
Contract Watcher: ethereum-contract-watcher



LESSONS LEARNED AND FUTURE WORK

Developing a production-ready blockchain product turned out to be far more complex than we initially expected from many perspectives: user experience, engineering and operations. When we started this project we were confident that we knew why we were using blockchain for our product and what benefits we would get, but as we gained more experience with it, we realised we had more question marks than answers or revelations.

Generally, the main advantages of Blockchain that we identified are:

- A tamper proof database which is very difficult to hack.
- A transparent database which everybody can verify independently for current and historical data.
- A distributed database which is resilient and can withstand many points of failure.
- A trusted database which include sophisticated authentication and authorisation protocols.

The challenges that we faced are:

- Communicating these benefits to customers and end users.
- Engineering a user friendly product which was secure and reliable.
- Operating the solution, monitoring it, specifically the complexity of buying and transferring Ether.

To summarise, Blockchain does introduce some new ideas and concepts and it holds many promises, but sometimes traditional, simpler solutions, although not as hyped, may be the best way to solve our problems.