DELOCK FUNCTIONAL SPECIFICATION

Project: Delock Rental System

Student Number: 14566803

Student Name: Mark McAdam

Course: CASE4

Table of contents

1. Introduction

A. Overview

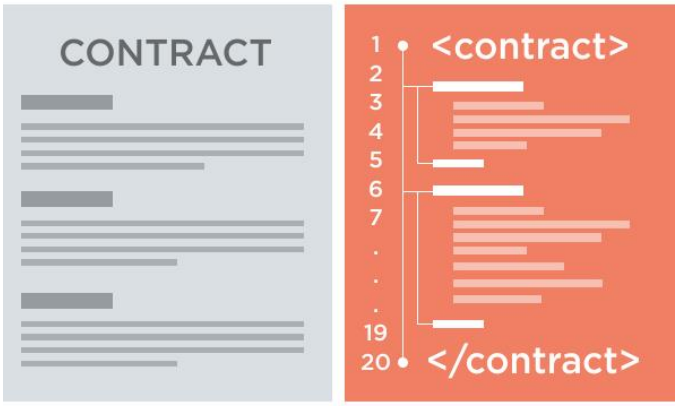
A new model for building large scale applications is emerging, making it possible for systems to reinvent traditional market behaviours. With the move towards the mainstream adoption of Bitcoin and cryptocurrencies in general, their underlying technology, Blockchain, has been the focus of a lot of speculation.

> *"The blockchain is an incorruptible digital ledger of economic transactions that can be programmed to record not just financial transactions but virtually everything of value."*
> *Don & Alex Tapscott, authors Blockchain Revolution (2016)*

Decentralized Applications have the potential to become standard, they could easily outperform major software corporations in terms of utility and user-base due to the emphasis on giving users incentives to maintain the network as well as flexibility, transparency, zero-downtime and immutability i.e. a third party cannot arbitrarily change a smart contract on the network.

The Delock system will be built on top of the Ethereum platform, Ethereum, like Bitcoin implements a blockchain to record transactions and essentially remove the need for trust as there is an immutable record of all activity on the network. The difference with the Ethereum network is its ability to program so called "Smart Contracts" or autonomous digital entities that can talk to and transact with each other without human interaction.

Smart Contracts can turn everyday objects into "Smart Objects" that can be rented, sold or shared on demand securely and without the middleman. It essentially gives everyday objects an identity and the ability to send and receive payments. This has the potential to significantly reduce transaction costs, increase security and simplify the user experience when renting items for both owners and renters.



B. Glossary

- **Blockchain** - is a continuously growing list of records, called *blocks*, which are linked and secured using cryptography. Each block typically contains a hash pointer as a link to a previous block, a timestamp and transaction data. By design, blockchains are inherently resistant to modification of the data.
- **Decentralization** – is the process of distributing or dispersing functions, powers, people or things away from a central location or authority.
- **DApp** – A Decentralized Application is a piece of software consisting of a user interface (UI) and a decentralized backend; typically making use of a blockchain and smart contracts.
- **Ethereum** – is an open-source, public, blockchain-based distributed computing platform featuring smart contract functionality
- **Infura** – provides secure, stable, and scalable Ethereum nodes.
- **Miner** - A node on the network which holds a copy of the blockchain and who performs hashing\mining operations to verify new transactions and create new blocks.
- **Smart Contracts** – is a computer protocol intended to facilitate, verify, or enforce the negotiation or performance of a contract
- **Solidity** — scripting language for smart contracts on the Ethereum Platform.
- **.sol** – Solidity File extension
- **ABI** – (Application Binary Interface) How to call functions and receive data from a smart contract.
- **Wallet** – is a software program that stores private and public keys and interacts with various blockchain.
- **Signed transaction** – When the private key stored in the user's wallet is used to verify a transaction.
- **Ethereum Android** – wallet for signing transactions generated by your DApp
- **Truffle** – development framework for testing, deploying, and interacting with smart contracts from the web
- **IPFS** – "InterPlanetary File System" is a protocol designed to create a permanent and decentralized method of storing and sharing files. It is a content-addressable, peer-to-peer hypermedia distribution protocol.
- **NFC** – Near Field Communication
- **Nonce** – an arbitrary number used only once in a cryptographic communication.
- **Fixed Asset** - assets which are purchased for long-term use and are not likely to be converted quickly into cash, such as land, buildings, and equipment.

2. General Description

A. System Functions

> "Collaborative Economy - Systems that unlock value from underused assets* by matching 'needs' and 'haves' in ways that bypass traditional intermediaries and distribution channels."

The Delock platform fits with the idea of a Collaborative Economy. It will provide a simple and straight-forward means of unlocking the value of unused or under-utilized assets ("idling capacity") whether it's for monetary or non-monetary benefits, in doing so it simultaneously puts a vast array of resources at the disposal of the public.

The general functionality of the system, these will be explained in more detail in later sections:

- Register / Login
- Create listing
- Manage Assets I.e. fees, availability etc.
- Manage Bookings I.e. Cancellation (renter)
- Confirm rentals (renter)
- Unlock objects
- Return Objects
- Debit payments from renter
- Send payments to owner

B. User Characteristics and Objectives

> *"Micro-preneurs - Empowering people to make and save money from their assets"*

Delock is aimed at anyone who has idle, under-used or other assets that have the potential to be turned into new revenue and profit streams. These assets have unused capacity, the owner can make this capacity available to others with peace of mind and security. The idea of trust between strangers allows anyone to rent their assets or to avail of these assets for rent.

On the other hand, the system would cater to those not interested in owning or who cannot afford to own certain fixed assets. It gives this demographic on-demand access to resources normally only available for purchase.

To use the application, users will have to have some familiarity with technology and the steps involved in downloading and using applications, a focus of the system would be accessibility and simplicity. The intent is to make this service available to everyone, including those who find technology intimidating and alien. This can be achieved by abstracting the technicalities of complex technologies such as blockchain and Ethereum away from the user and leaving them with a streamlined means of interacting with the system.

One hurdle that may be an issue for some users will be the creation of an Ethereum wallet. For most of the population, cryptocurrencies and decentralized technology is a foreign concept and this may inhibit adoption or understanding of the Delock system early on. Mainstream adoption of these technologies is not far from a reality and more and more people are becoming attuned to how these systems function and the benefits they could provide.

C. Operational Scenarios

User Sign-up

- User downloads Delock from App Store.
- Delock will check for a valid installation of the Ethereum Android Wallet and prompt user to install it if none exists.
- User links existing Ethereum account using account address.
- On signup, user details are collected into a JSON object.
- This object is stored into a JSON file and the file is published to IPFS
- Get the files hash i.e. location on IPFS
- Initialise a smart contract that associates the user's Ethereum account with the IPFS file hash.

User Validation

- User opens the application
- Ethereum Android provides the active users account
- Read from the smart contract to get associated IPFS hash
- Retrieve file from IPFS
- Read JSON object
- Extract contents and display to user.

New Object Profile

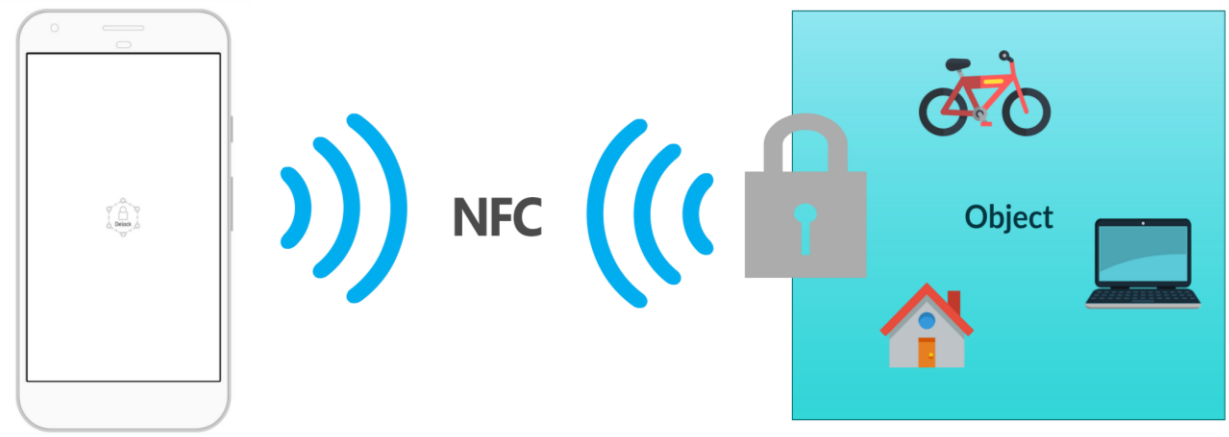
- Owner opens the app
- They are presented with a screen with two options in the form of
    - Rent stuff from others
    - Create \ View my items
- Owner clicks the latter and then clicks an option to create new profile
- Owner enters all required details about the object, namely:
    - Title – “Sports Bike for rent”
    - Images and Pictures of the object
    - Description – “Spotless, fast and sturdy”

- Availability – Owner defines days or periods the object is unavailable.
- Deposit price - €25
- Rental price - €2 per hour
- Condition – “Like new, 3 months old”
- Contact Details

- Owner selects “Create Profile”
- A smart contract is generated that specifies the requirements to rent the object i.e. “deposit amount must be met”, “It cannot already be in use” etc.
- Information about the object not needed by the smart contract such as title, images and description are stored into a JSON file and sent to IPFS.

(OWNER) Programming the lock

- Once the owner has created a profile for the object, they need to activate the lock.
- To do this they must enable NFC, click a button “Program Lock” and hold their device against the lock.
- The lock will receive the address of the relevant smart contract on the blockchain and store it.
- The user then locks the object with the physical lock.
- Once this has completed successfully, the smart contract can be “activated”.



- (RENTER) Browsing for objects to rent
- The renter opens Delock
- They have options to
    - Use the search bar at the top of the screen
        - Using keywords e.g. “Bike”
    - Filter along certain parameters and details
        - “Condition = New”
        - “Rental price = €2.50 hr”
    - Choose a category thumbnail and expand it to show a group of similar objects, they can then scroll through these.
- Renter can click any of the object thumbnails to expand them and view a more detailed profile of the object
- and choose an option to book it.

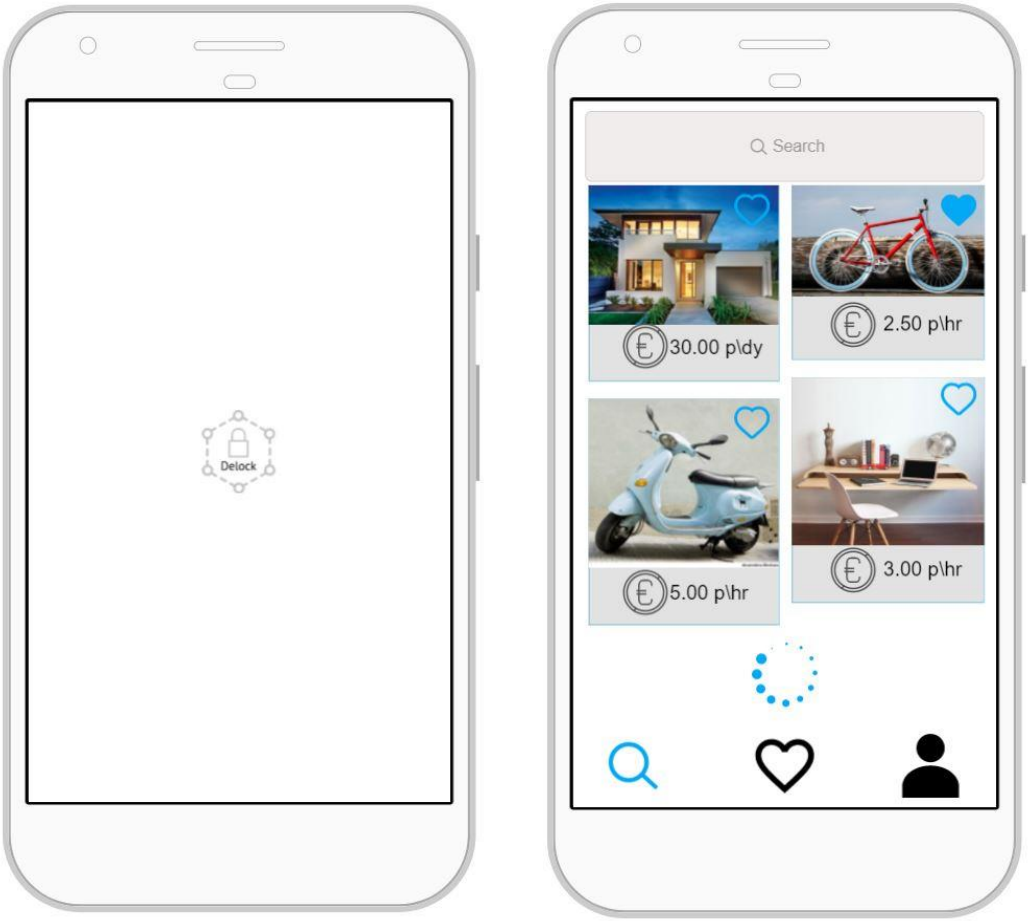


Fig. Loading screen and Browse\Search screen mocks

(OWNER) Managing an object

- Owner opens the app and clicks the “My Stuff” icon represented by a person silhouette below.
- Items can be active (available to rent) or Inactive (unavailable).
- The owner can click on any of these thumbnails to see the status of that object, for example:
    - Status – “Available” or “Rented by Joe Bloggs 3 hours ago”
    - Option to disable the object and stop it being rented, only if not currently rented.
    - Owner can request the object be returned early.
    - Owner can select dates and times at which the object is unavailable for rental.
    - Owner can update object info and details.
    - If the owner decides to disable an object then its connection to a lock will be reset and when re-activated the owner must complete the “Program Lock” steps again.

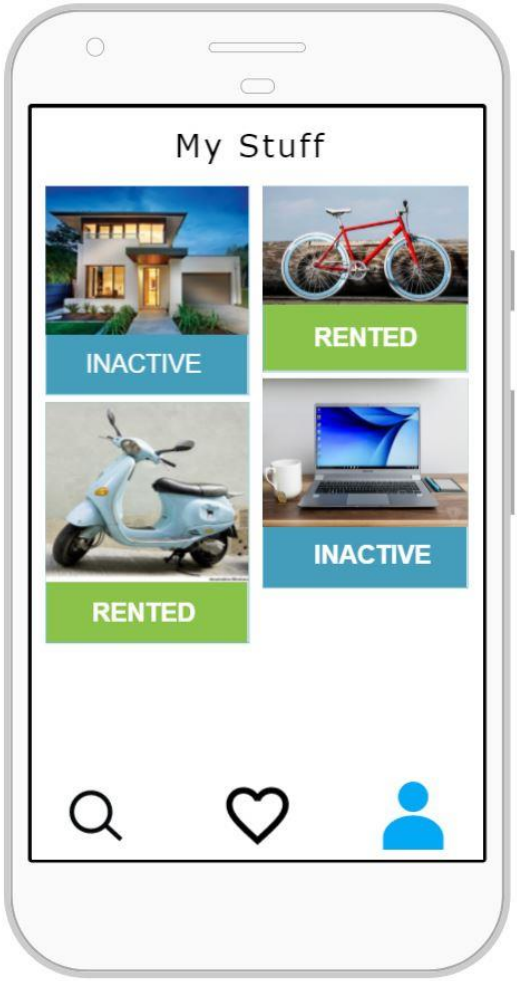


Fig. “My Stuff” screen showing owners items and status mock

(RENTER) Renting an object

- Renter opens the browsing screen in Delock
- Renter navigates to an object they wish to rent, they click the thumbnail
- They check the object's availability
- If the object is available then they can choose to rent it
- They are taken to the confirmation screen and can complete the transaction and secure the booking.
- This object will be added to the renter's home screen as "Pending" until they unlock the object and then it will be "Rented".
- Bookings are valid for one hour until the object is made available again.

(RENTER) Unlocking an object

- The renter has completed the previous step and has a valid booking
- They arrive to the object's location, if their booking is still valid then they can continue to the next steps, otherwise they must re-book.
- The renter must open the Delock app and turn on their NFC.
- Under "My Rentals" they select the object and click an option to "Unlock"
- They then hold their device against the lock on the object
- The lock will authenticate the renter's device and if everything is successful it will disengage.

(RENTER) Using the lock between initial rental and return

- Once the initial Unlock step shown previously has been completed, the lock will switch "modes", this means that the renters device can open/close the lock as many times as they please.

(RENTER) Returning the object

- The renter must return the object to an agreed location or they can leave it within a radius defined by the owner.
- The renter locks the object, they then open the Delock app and turn on their NFC.
- Under "My Rentals", they select the object and an option to "Return".
- They then hold their device to the lock and this completes the process.
- The renter can no longer open the lock and payment is calculated and sent to the owner.
- The renters deposit is also returned to them.

D. Constraints

Blockchain is still a technology in its adolescents, there is a very active community but there are still a lot of unexplored applications of the technology and this could be a drawback when building this system.

A major constraint of the system is cellular infrastructure in the area where the system is being used, renters need to have speedy connections to the internet to allow communication with the blockchain when they arrive to unlock or to return an object. Slow connections could significantly slow down the speed at which they are able to be on their way.

Down-time should not be an issue due to the decentralized nature of the system, if all but one node in the network fails, the system will still be live. Of course, it would be slowed down significantly.

Ideally, all the locks would act as the nodes in the network and when a new transaction occurs all the locks would communicate with each other and verify the validity of the transaction. Due to the expense of having every lock enabled with LTE, this is not a viable option now.

3. Functional Requirements

Registration \ Login – As a user I should be able to register and login using my Ethereum address and access all the applications functionality.

User Requirements

- Valid Ethereum Address

System Requirements

- Connection to Ethereum platform

Acceptance Test - Registration \ Logging on

Notifications – As a user, I should be notified of any changes for example: 'changes to my assets status' or 'messages between renters or owners'.

System Requirements

- Android Notification Class
- Push notifications on update

Acceptance Test – Receive Notifications

Create Profile/Listing - As an owner, I should be able to create profiles/listings for personal assets as "For rent" after providing the required parameters and relevant details about the asset.

User Requirements

- Asset to list and details

System Requirements

- New smart contract on Ethereum network
- New storage address on IPFS

Acceptance Test – Create new listing

Edit Profile/Listing - As an owner, I should be able to make changes to the profiles/listings of assets that I have up for rent.

System Requirements

- Users Ethereum address
- IPFS address obtained from Ethereum

Acceptance Test – Edit listing

View Profile/Listing - As a user, I should be able to view the status my assets for rent and also assets I have rented.

System Requirements

- Connection to IPFS

Acceptance Test – View Details

Submit Request - As an owner, I should be able to request the early return some asset that I own if it is currently rented.

User Requirements

- Message to send

System Requirements

- Owners Ethereum Address
- Renters Ethereum Address

Acceptance Test – Submit request

Historical Transactions – As an owner, I should be able to view of list for each of my assets rental history.

User Requirements

- Valid Ethereum address

System Requirements

- Connection to IPFS and Ethereum network

Acceptance Test – View History

Payment – As a user, I should be able to receive/send payments

User Requirements

- Valid Ethereum Address
- Sufficient funds

System Requirements

- Total to debit
- Connection to IPFS and Ethereum network

Acceptance Test – Receive payment

Browse – As a renter, I should be able to browse/search a collection of all available objects.

System Requirements

- Connection to IPFS

Acceptance Test – Browse listings

Booking – As a renter, I should be able to place a booking on an object that will be valid for one hour from the time of confirmation.

User Requirements

- Sufficient funds for Deposit

System Requirements

- Connection to Ethereum platform

Acceptance Test – Confirm booking

Cancellation – As a renter, I should be able to cancel a booking before it naturally expires.

System Requirements

- Connection to IPFS - (transaction has not yet been processed on Ethereum, only the object's status has been changed to “Booked” on IPFS)

Acceptance Test – Confirm cancellation

Favourites – As a renter, I should be able to add objects to a “favourite folder”.

System Requirements

- Connection to IPFS

Acceptance Test – Add listings to favourites

Unlock – As a renter, I should be able to press the unlock button with NFC enabled and have the physical lock disengage.

User Requirements

- Confirmed Booking
- Sufficient funds for deposit
- NFC enabled

System Requirements

- Connection to IPFS and Ethereum network

Acceptance Test – Lock Disengages

Return – As a renter, I should be able to press the return button with NFC enabled to return the object when I have finished using it.

User Requirements

- Rented object
- Sufficient funds for payment
- NFC enabled

System Requirements

- Connection to IPFS and Ethereum network

Acceptance Test – Renter cannot open lock afterwards

Review – As a renter, I should be able to leave a review or rating of the object that I rented to help other makes informed decisions when renting.

User Requirements

- Ethereum address - Must have record of renting this object

System Requirements

- Connection to IPFS and Ethereum network

Acceptance Test – View Details

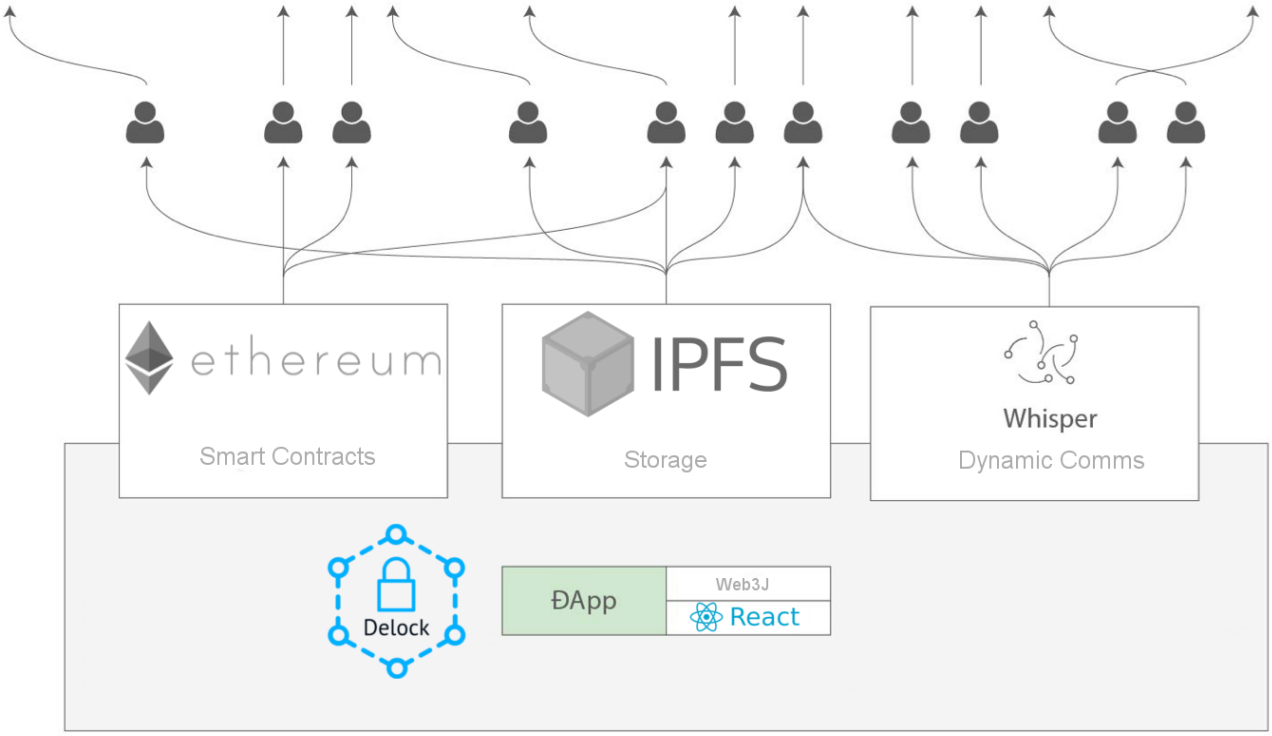
Data Requirements

- Users Data
    - Ethereum Address
        - Transaction History
    - IPFS Address
        - Name
        - Contact Details
        - Listed Items
        - Active Rentals
- Listings Data
    - Ethereum Address
        - Smart Contract
        - Transaction history
    - IPFS Address
        - Images
        - Title
        - Description
        - Status
        - Reviews
        - Misc. details

Hardware Requirements

- Arduino
- Arduino NFC Module
- Mechanised Lock System
- NFC equipped Smartphone

4. System Architecture



Above we can see a high-level view of the system and the different components and third-party systems that will make it up.

The grey box at the bottom of the diagram represents the Delock android application itself, the smaller white boxes overlapping are the third-party systems and protocols being used by the applications. Above this is a representation of how the system is distributed among many peers.

Delock Application

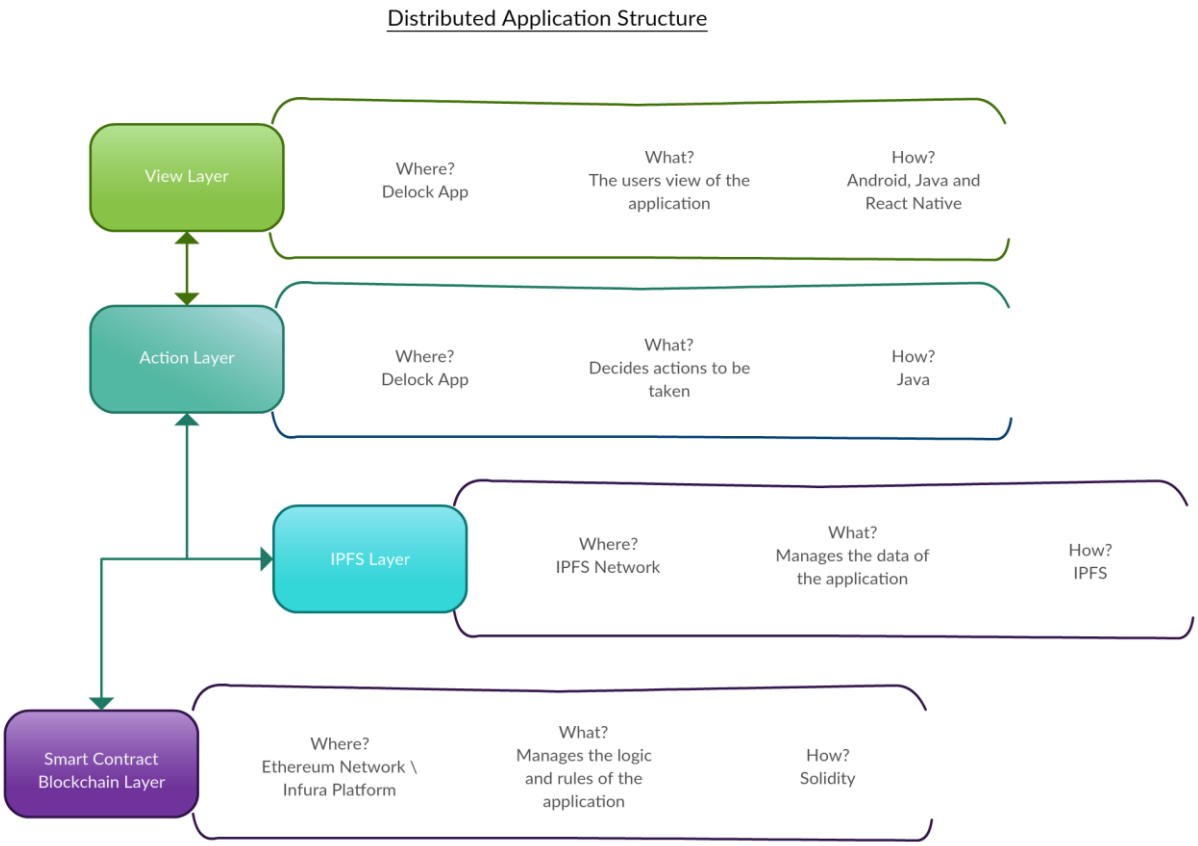
The main application will run solely on the Android Operating System as it requires access to the NFC functionality of the host device and this is not available on IOS. The application will be built using a combination of React Native and Web3J library.

React Native is a library from Facebook, it is efficient for creating performant IOS or Android applications meaning the application could easily be ported to IOS should Apple make the NFC functionality available to developers in the future. React is very much geared towards simplistic and stylish user interfaces, this will help to make the experience of using the Delock application enjoyable and intuitive.

Web3J is a library for interacting with the Ethereum network from a mobile device, the application will need extra configuration to work with the Infura (Ethereum in diagram above) implementation of an Ethereum Network.

Ethereum / Infura Platform

The first of the third-party systems depicted above is the Ethereum Network, the Delock application will use the Infura service for accessing the network. The application will be developed and tested using a test network provided by Infura. The network consists of a multitude of Ethereum nodes meaning each has a copy of the blockchain and smart contracts, when a user performs a transaction in the Delock application it will contact these nodes and the smart contract code will execute and decide the validity of the transaction.



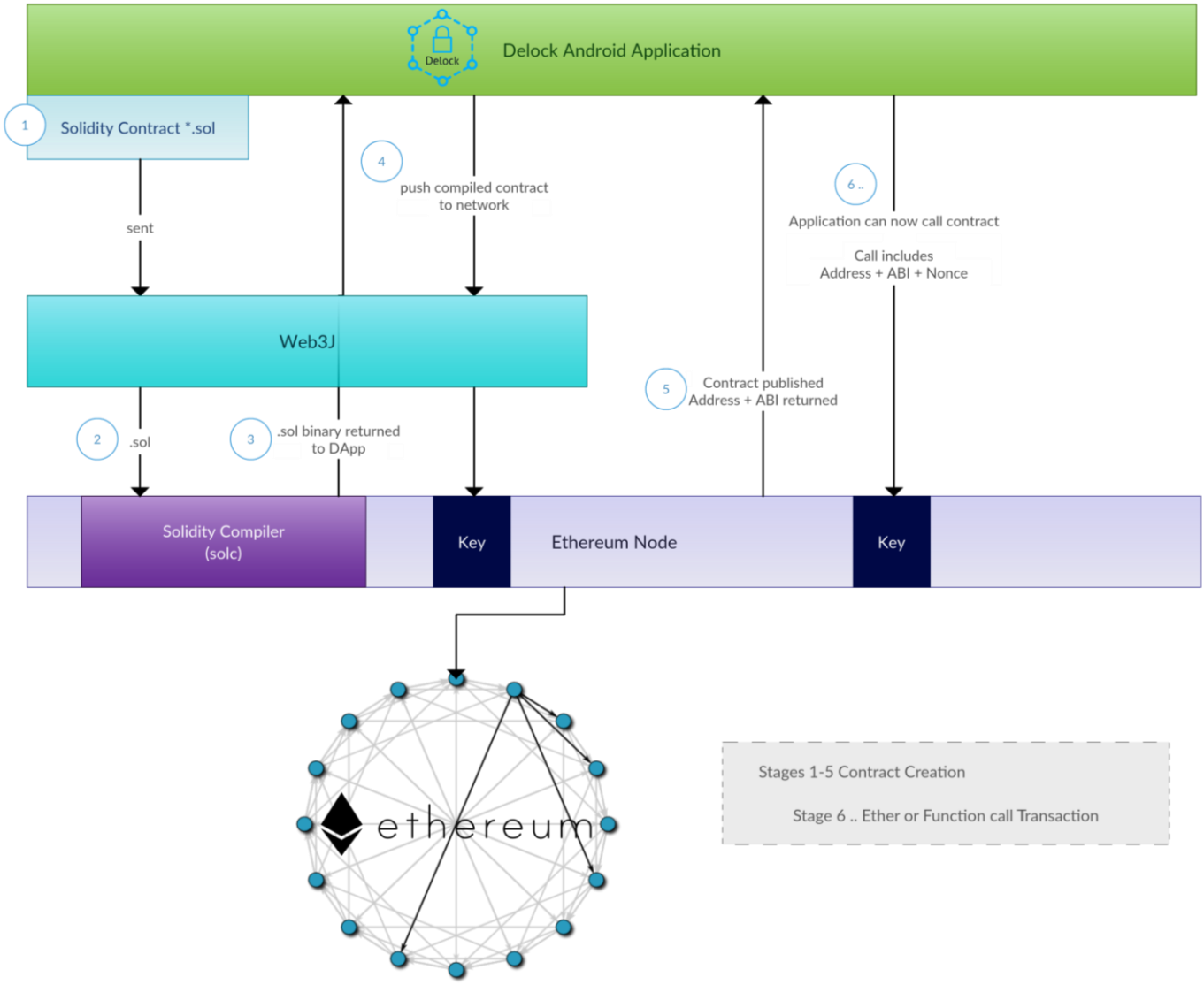
5. High-Level Design

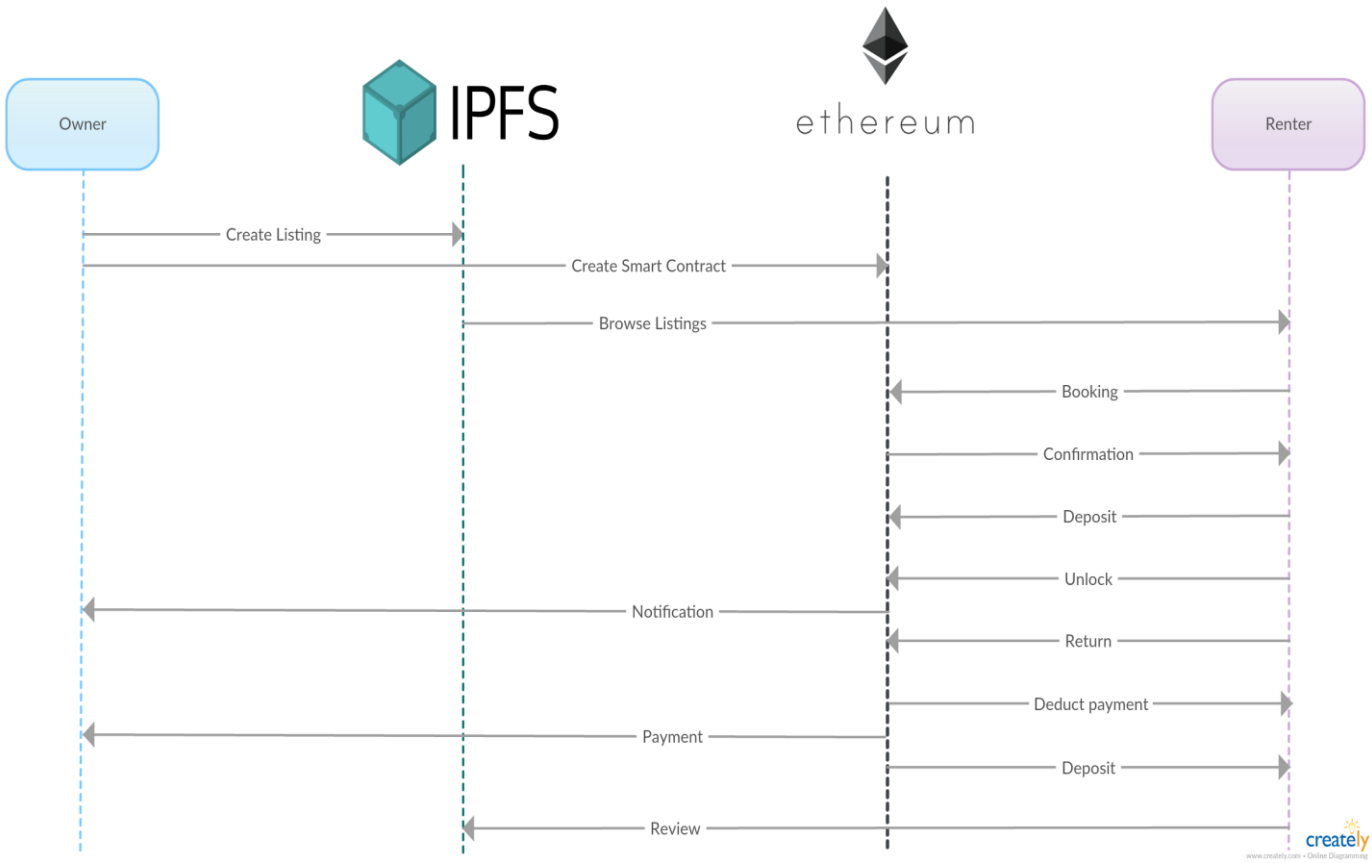
Below we can see the stages involved in the creation and propagation of a smart contract to the network using a private key.

The contract is generated in the Solidity language (.sol) by the Delock Application, it is then sent via Web3J to the Ethereum node for compilation.

Once compiled, the binary is sent back to the application, Web3J is then used to publish the contract to the node with the user's unique private key for verification. We can see that the node is connected to the network and by stage 5 the node has published the contract to the network and will proceed to return the reference details to the application. These consist of the contracts address on the network and an ABI (Application Binary Interface) which defines how to call functions and get data back from the contract.

With this the Application can now call the contract using the users private key. A "nonce" prevents publishing of duplicate transactions and extends the lifespan of a key.



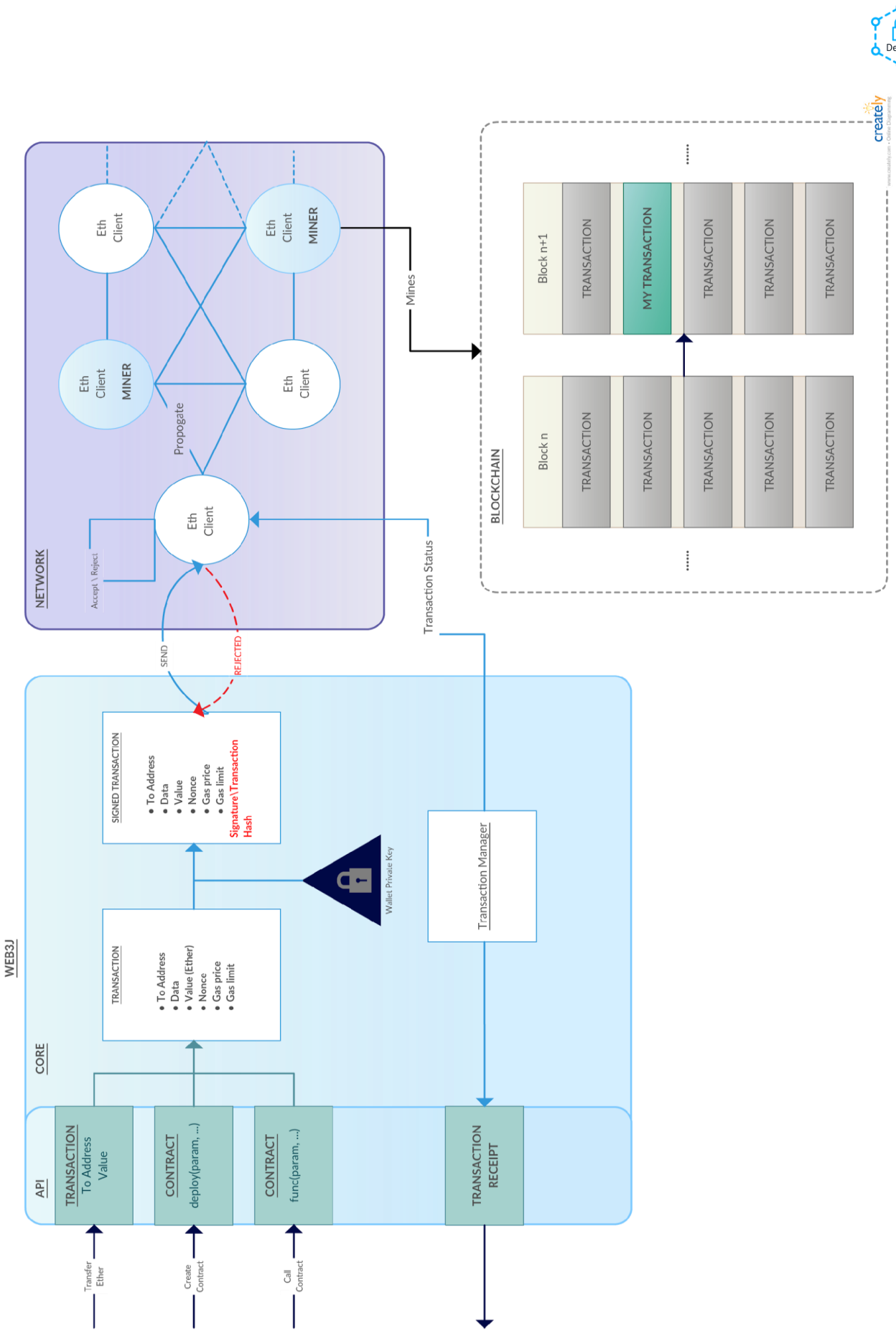


On the following page, we can see a high-level view of the interaction between components and an illustration of data flow within the system. From left to right we can see calls to the Web3J API such as “Create Contract”, these are handled by the Web3J API and passed on to the Ethereum Network.

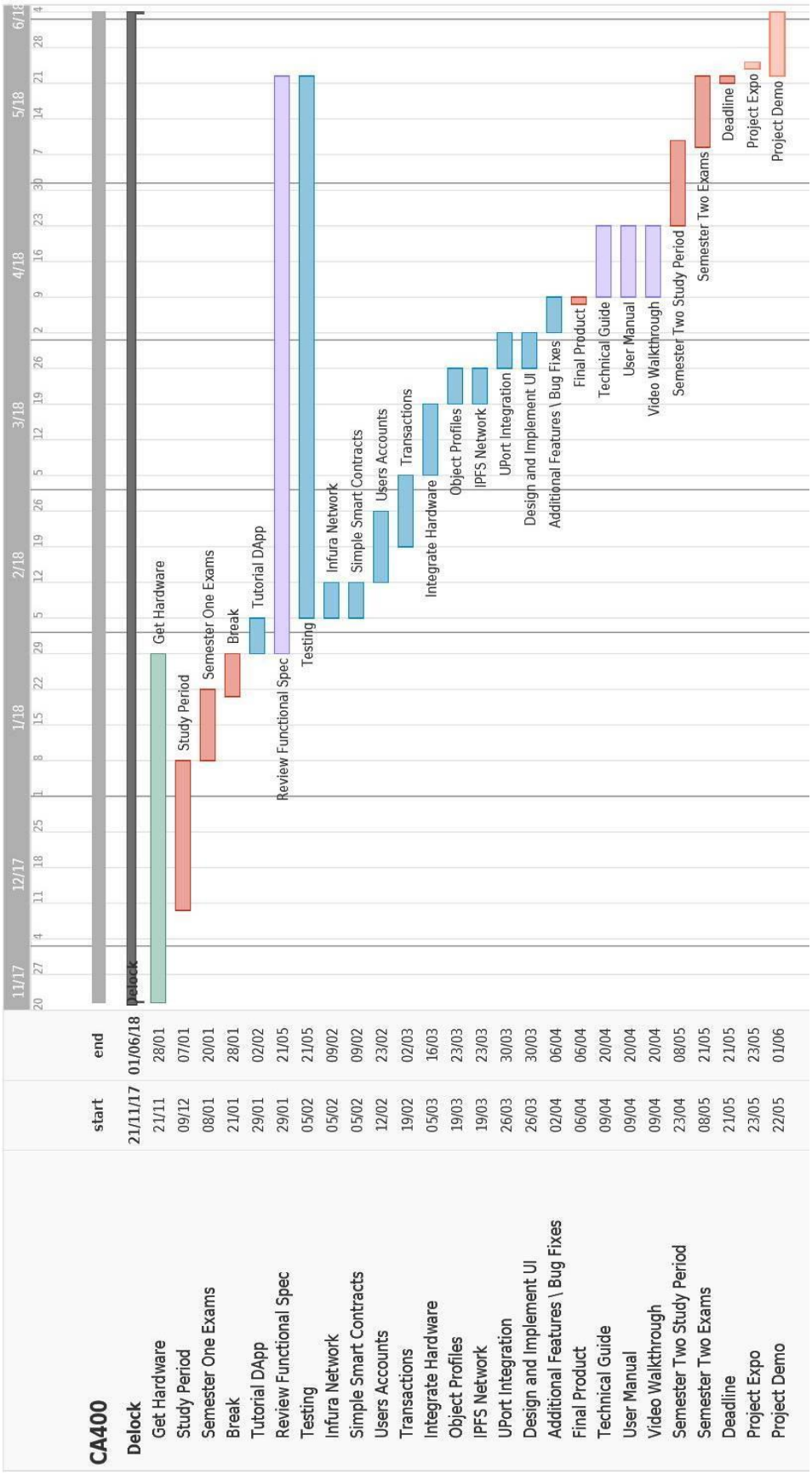
A new transaction is created by Web3J and the user's Ethereum wallet signs the transaction with the user’s unique private key. The signed transaction is then sent to the Ethereum network where the nodes decide between themselves whether this is a valid transaction and then accept it or reject it.

Once accepted, a mining node will then bundle this new transaction with a multitude of others and hash a new block to be added to the blockchain i.e. (mine a new block), this change will be propagated to every other node on the network hosting a copy of the blockchain.

Not every node in this network will hold a copy of the blockchain, this is because a lot of the nodes will be user’s mobile devices meaning the storage requirements make it impractical.


Delock
creately
NETWORK
Eth Client
Eth Client MINER
Eth Client MINER
Eth Client
Eth Client
Propogate
Accept \ Reject
Mines
BLOCKCHAIN
Block n
Block n+1
TRANSACTION
MY TRANSACTION
WEB3J
CORE
API
TRANSACTION To Address Value
CONTRACT deploy(param, ...)
CONTRACT func(param, ...)
TRANSACTION RECEIPT
TRANSACTION
• To Address
• Data
• Value (Ether)
• Nonce
• Gas price
• Gas limit
SIGNED TRANSACTION
• To Address
• Data
• Value
• Nonce
• Gas price
• Gas limit
Signature \ Transaction Hash
Wallet Private Key
Transaction Manager
SEND
REJECTED
Transaction Status
Transfer Ether
Create Contract
Call Contract

6. Preliminary Schedule



7. Appendices

Decentralized Applications (DApps) - https://blockchainhub.net/decentralized-applications-dapps/

https://coinsutra.com/dapps-decentralized-applications/

Ethereum Project - https://www.ethereum.org/

Smart Contracts - https://bitsonblocks.net/2016/02/01/a-gentle-introduction-to-smart-contracts/