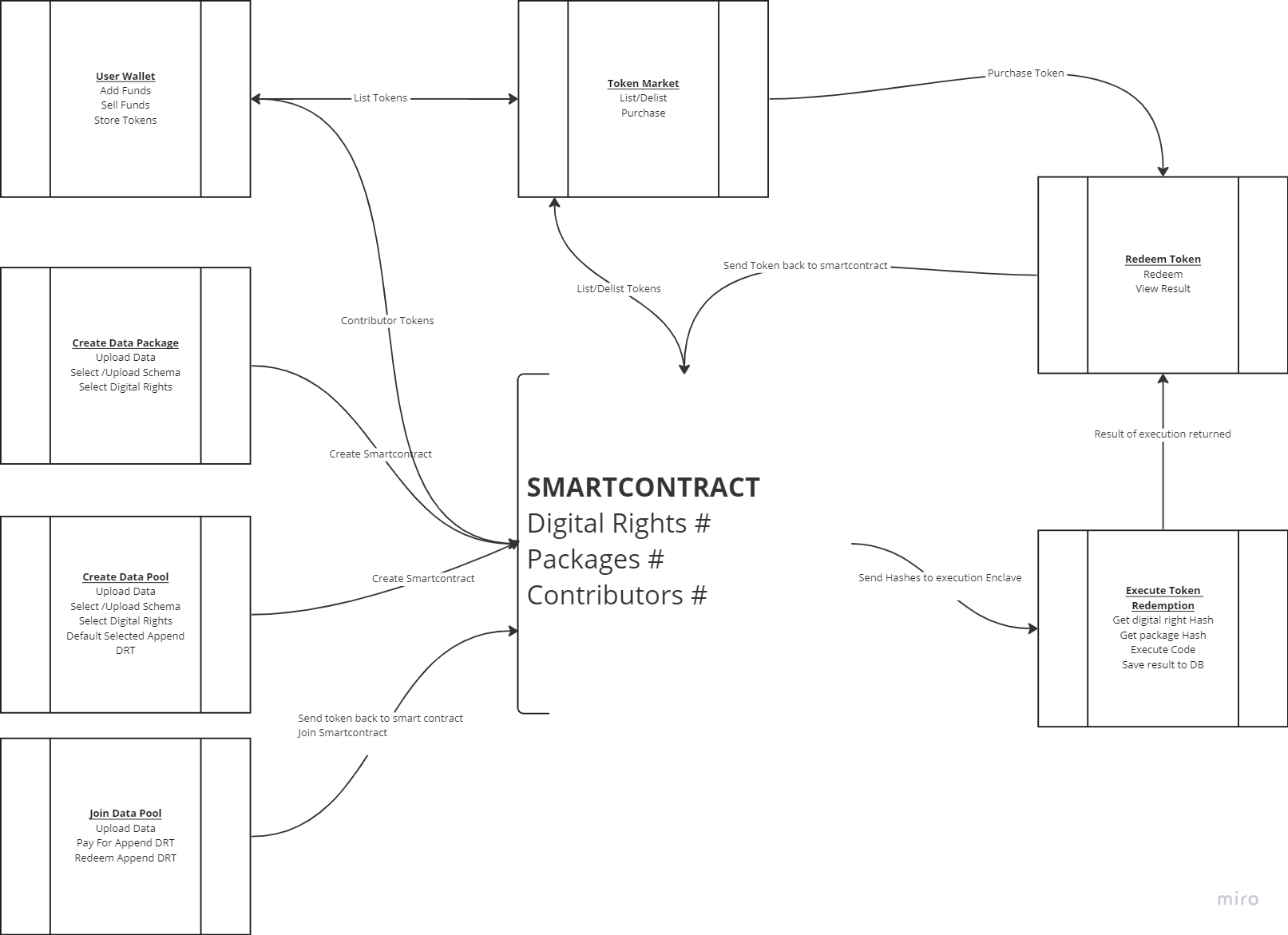
**Nautilus Trusted Compute MVP**

**Research Notes**

Jan 7, 2023



03 Feb Notes

**Smart Contract**

* Determine if test network has all smart contract functionality
* Smart contract must enforce 5% royalty fee for when DRT is bought off the smart contract (primary market), currently the smart contract only enforces the 5% royalties for the secondary market. The royalty fee must be enforced on top of the sale price.

# User Requirements

### 0.0 DRT Definition

A Digital Rights Token (DRT) is defined by:

1. Unique name (40 Char?)
2. Short description (150 Char?)
3. Unique data pool address
4. Unique reference to code

*Comment: Each DRT created by the smart contract has the following information:*

* *Asset ID*
* *DRT name*
* *DRT Unit name*
* *URL of code binary*
* *Hash of code binary*
* *Supply of tokens to create for this DRT*
* *Exchange Price of DRT*
* *Creator of the asset i.e. the smart contract address*

### 0.1 DRT States

What state a DRT is in, depends on its relation to the user and the user’s data pools. Specifically, DRTs are in one of four states:

1. Defined  
   These are DRTs that have been defined by *any* user, but have not been created for a given pool. Manage Data Pools allows the user to select a defined DRT and create it for this pool.
2. Created  
   These DRTs have been defined by any user and have been issued by a user during Create Data Pool or Manage Data Pools. A user can Change Price of a created DRT, even if it has not been issued. Once a created DRT is purchased by a user, its status for the creator changes to Issued. A user can decide to Unlist a created (but not yet issued) DRT, which unlists it on our platform and removes it from the smart contract.
3. Issued / Purchased  
   If the user is a data creator, i.e. has set up a data pool, then she can create DRTs (for the specific pool). A DRT that is purchased by another user is Issued. If a user buys a DRT of another user, it shows as Purchased (e.g. the AppendDRT required to join another user’s data pool). A user can Sell and Redeem a purchased DRT and Change Price of the DRT (on the Nautilus platform, not sure if we can enforce this on other platforms).
4. Redeemed  
   A DRT that has been purchased can be redeemed. Once it has been redeemed, the state of a DRT cannot be changed anymore.

### 1.0 User

The term “user” is generic and includes data and code creators, data contributors, and analysts.

A user can register on <https://ntls.io> and Sign Up to create a new profile. During sign up, we collect a username and password, as well as an email address and phone number (which we verify to complete the signup).

Once a user returns to the web app, she can Login. A user who is logged in sees a button to go to their Profile. If this option is selected, users can manage their account settings and personal details (for now, only username, password, email, and phone number). They can use the app to set up a new unique wallet address for their numeraire (e.g. $Algo), Contributor Tokens and digital rights tokens. For this, we set up a new Algorand wallet for users using the Wallet library and show the wallet address in the personal information page. We also collect information about a user’s external Algorand wallet, used to pay out later. We aim to provide functionality so that users can fund their newly created wallet using ~~AlchemyPay~~, TransAk, or BanXA pay.

The user’s password is also used to manage the private key associated with the user’s wallet (set up via the wallet app) and handled within an enclave. Users provide an email address, which we use to provide a key recovery option. We aim to also verify a user’s phone number and implement 2FA for improved security. 2FA will eventually be used for a user to create a pool, change the status of a digital right (under Manage Digital Rights), list or delist contributor tokens, create a new digital right, join an existing data pool, eventually to delete a pool, and to delete a user profile.

Eventually, we will also collect personal information necessary for KYC and give users the option to verify their personal details (name, date of birth, passport number) by going through a KYC process.

Users can Withdraw their funds to their personal Algorand wallet by authorizing transactions from their numeraire wallet to an Algorand wallet address of their choice. Eventually, we allow users to withdraw funds into their preferred fiat wallet using alchemy pay.

Users can delete their account.

### 1.1 Data creator

A data creator is a user. Using the vault (web) app, a data creator can create a data pool. For this, she logs in to her user account. Once the data creator is logged in, she sees a list of all her data pools below a button to create a new pool. She also sees a dashboard with meaningful summary graphics about her data pools, created and assigned (issued) DRTs, Contributor Tokens, funds ($Algo) received in their numeraire wallet, and funds paid out (to her preferred external wallet).

Once a data creator clicks to Create Data Pool, she is guided through a four-plus-one step process. First, she defines a data schema for the pool according to a standard defined and published by Nautilus. Second, she selects the file containing one or multiple rows of data conforming to the schema. Third, the user selects how many Contributor Tokens—which are DRTs—she wants to receive for the data. Initially, users can have only one wallet, but we aim to allow them to have multiple wallets and to select a preferred wallet for each pool. Fourth, the data creator selects from a list of data rights that are available for the pool. It is also possible to skip this step and add data rights later. Data rights are initially defined by Nautilus, but we aim to have functionality for users to create new data rights. And fifth, the data is uploaded.

For each existing pool in her list, the data creator is presented with an option to Manage Digital Rights. This allows the data creator to TODO.

Create New Rights, See Existing Rights, and Manage Data Pools.

When Create New Rights is chosen, data creators are guided through a process where they can select new digital rights from a list of possible rights (including the AppendDRT, which is provided by Nautilus) and the data creator selects a supply and price schedule for each new right. Initially, the supply and price are static and fixed ex ante (i.e. before issuance), but we aim to allow users to adjust both schedules ex post and eventually to define a form of token bonding curve to manage the price of a DRT.

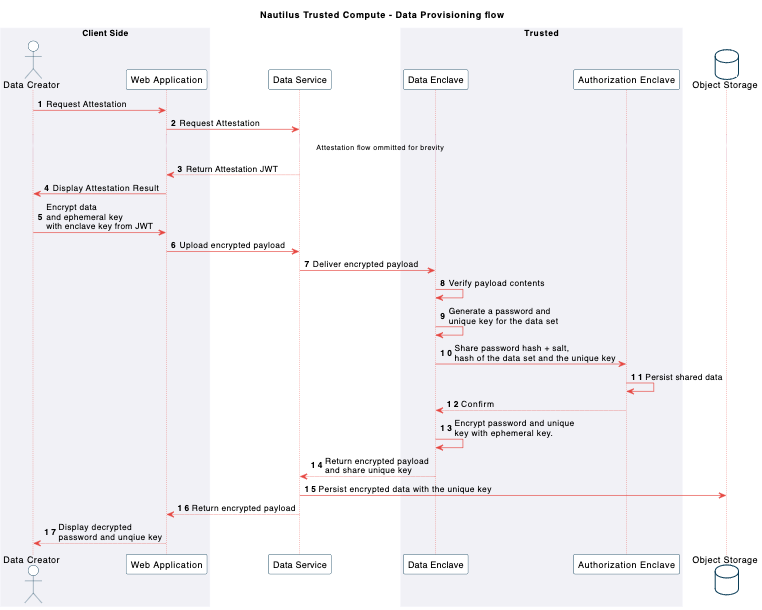
Rights are created by instantiating the data pool and associated smart contract. They are assigned by issuing Digital Rights Tokens via an interface with an Algorand smart contract.

When See Existing Rights is selected, the data creator is presented with a list of all existing rights and a meaningful overview of how many of each have been issued (e.g. graphical), at what price (in $Algo) and how much has been paid to their numeraire wallet.

When a pool’s Manage Contributor Tokens is selected, the data creator is shown a meaningful overview of the pool’s Contributor Tokens, including how many tokens the data creator holds and how many tokens have been created in total. The data creator can Sell Contributor Tokens. If selected, a token, or a set of tokens is internally marked as for sale. We also aim to have functionality to allow data creators to de-list Contributor Tokens that have been marked for sale.

When Delete Data Pool is selected, the platform checks if all Contributor Tokens have been returned to the platform. If that is not the case, the platform tells the data creator how many Contributor Tokens are still outstanding. The data creator is then asked whether she wants to redeem Contributor Tokens and a workflow is started that allows the data creator to send Contributor Tokens to the smart contract managing the data pool. Once all Contributor Tokens have been redeemed, the data creator can actually delete the pool.

Data creators receive a proportion of the funds received from selling DRTs (including the AppendDRT) and Contributor Tokens into their numeraire wallet. They can choose to Pay Out Funds to a preferred wallet. We aim to provide 2FA functionality for this step.



### 

### 1.2 Data contributor

A data contributor is a user. Once logged in, the user sees a *list of existing data pools* (i.e. pools with available AppendDRTs) below the dashboard described under 1.1. and can select Join Existing Data Pool.

Once selected, a workflow with four steps starts. First, the user must purchase an AppendDRT using funds in her numeraire wallet. Users are notified if they have insufficient funds and the flow terminates. Second, once the user purchases an AppendDRT, it is transferred to the user’s wallet. Third, the AppendDRT is transferred from the user’s wallet to the smart contract managing the data pool. This step is necessary so the smart contract has proof that the AppendDRT was sent by its new owner. Fourth, once the smart contract has received the AppendDRT, the data upload workflow starts.

In this workflow, the user is asked to upload the data and a matching data schema file. We aim that the system verifies that the data corresponds to the schema and that the schema corresponds exactly to the data schema of the pool. Once the files are uploaded, the user receives a Contributor Token and a confirmation message that the contribution was successful.

If an analyst purchases a DRT for the data pool, data contributors receive royalties into her numeraire wallet proportional to the sample size of their data.

### 1.3 Code creator

For code to be referenced in a DRT, it must be hosted in a publicly accessible repository which satisfies three conditions. First, it contains the actual source code and a suitable license (e.g. GPLv3). Second, it contains the corresponding WASM binary. And third, it provides instructions for how to reproduce the WASM binary from the source code.

Initially, Nautilus provides sample code in our github repository.

We aim that a user can Create Code by claiming ownership of a public github repository.

We aim to allow users to host their source code and WASM binary directly on our platform and issue DRTs that codify how others can use their code. For this, a user can start a flow that creates a repository which satisfies the conditions outlined above.

Eventually, we will give users the ability to become code creators by linking to their existing github accounts and creating corresponding DRTs. We then also allow users to become code contributors.

### 1.4 Analyst

A user becomes an analyst if she selects an existing data pool and chooses Purchase DRT. The purchase of a DRT is a simple process where the user must approve payment from her numeraire wallet to the smart contract managing the DRTs. Once payment is completed, this smart contract will send the user the DRT to her wallet.

Users who hold DRTs can choose to redeem them with our Code Execution Service. For this, the user selects Redeem DRT, which launches the code execution in the background. When code is executed, the smart contract managing the data pool used in the code execution periodically bills the analyst’s numeraire wallet. Once the code execution is completed, the user receives a notification that the result of the computation is available for download from their user dashboard as long as they are able to pay for the data storage from their numeraire wallet.

Analysts can also see a dashboard summarizing their usage and payments history. Furthermore, analysts can delete the result of code execution.

# Notes on Code Execution System

**Executable Code**

* Code will be public for an MVP but in the future we should allow for proprietary code, and a private auditing system for non-public code.
  + Code and data are not fundamentally different. Post MVP, we will cater for code creators like we now do for data creators.
  + This necessitates a mechanism to match DataDRTs and CodeDRTs so that analysts have it easy to find matching code and data.
* Data creator, when creating a pool can reference an executable. Both the code and data creators must specify the price or percentage of their property.
* Restrict or allow users or types of users for both code and data - e.g. “my code can only be used by nonprofits, or only by verified users”. This would need to be either self-enforced or done via KYC or an external certification provider.
* A code creator would create a smart contract. How do these stand in relation to the data contracts?
  + Set up code as a “data pool” with ourselves as the code author, and, for the MVP, disallow new code creation.
  + Or we can provide a set of executables that are internal to the system and
* How is code provided to the Vault? The user can upload a binary. Better yet is to have the user provide a URL to a repository with both a binary and the code. The DRT contains a hash of the binary and a URL to the binary and a reference to the smart contract that created the DRT which represents the data pool the DRT is allowed to run on.
* After a DRT is redeemed and authenticated by the backend, requesting by a particular dataset, the code and data are retrieved and unsealed. We need to still develop Rust code to

Flow of data analyst

1. Log in to Vault app that is loaded with a positive Algo balance
2. Browse available datasets, select a dataset and a DRT from the list of issued DRTs for that code.
3. Select a DRT from the list of owned DRTs. Push button to redeem that DRT.
4. Wait for code execution to complete.
5. List of results are shown and a file is available to download containing the output of the execution.

Flow of data pool creator

1. …
2. Select from list of available DRTs

**Computing Cost**

How do we account for the costs of compute resources used to store datasets and run analysis code? It is impossible to deterministically benchmark code for any dataset, and would be difficult to estimate. It then must be the responsibility of the user running execution code to adequately estimate the funds needed to complete an analysis. The most straightforward solution to ensure payment is to require a pre-paid escrow account for each analysis execution. The following conditions apply:

* Enclave tells smart contract how long code execution is running and what amount to draw from the dedicated user wallet;
* When smart contract is unsuccessful in drawing funds, there is a failed transaction, this needs to be monitored by the enclave;
  + This is done as follows: When execution enclave is set up, it is told by the smart contract the amount that has been pre-funded;
  + Enclave can then check from the system what the time is and thereby identify how long it runs; [Tshepang?]
  + This gives the enclave a measure of cost; Once the execution is completed, the enclave tells the smart contract to draw the funds from the escrow account;
  + If the enclave thinks it might run out of funds, it’ll stop and tell the smart contract to draw all available funds from the escrow, the execution may still fail or return partial results;
* User needs to pre-fund account managed by Nautilus; Perhaps set up new smart contract as escrow account;

TODO:

* Research the way to measure computing resources in the enclave. Determine how to measure CPU-equivalent time. Is it possible to measure time or cycles inside the enclave or can we measure time from outside the enclave? -> JP will be able to tell us the cost per hour of a basic Azure enclave; Joe to check; (~$0.11/minute)
  + Implementation: Write external demon that checks how long the enclave has been running and tells the smart contract when to charge user numeraire wallet next 30 minute pre-funded;
* ATTESTATION RESEARCH:Enclave reports truthfully the correct binary code is being run in the enclave. -> Joe/Co/JP details
  + Include hash of binary in enclave report..?
* Decide how pools are named

# Notes on Authentication and Funding

As usual we need to maximize both the ease of use and security. This section discusses account creation, setting up an algorand wallet, and funding a wallet. These are the options for user wallet creation:

1. Generate a Nautilus wallet for each Vault account.
2. Allow users to use their own Algorand wallet or
3. Create a wallet on behalf of users but leave private keys on the client computer. Provide push-button functionality on the client for all vault operations.

These are the options for funding wallets and for on-ramping:

1. Refer users to a crypto exchange such as Coinbase. Provide instructions on how to fund a wallet.
2. Use a payment platform in combination with a crypto exchange to build a tool for users to purchase services from Nautilus, by way of Algo.

Eventually we would like a rich offering with all wallet options: the nautilus wallet and a self-generated wallet, the ability to fund balances through our app, or to fund a wallet on an external exchange. For an MVP we must prioritize all work. We will first develop a Vault app that creates a Nautilus wallet for all new users and uses the existing wallet backend to sign smart contract transactions. This provides the safety and ease-of-use benefits of the Nautilus wallet with full integration with the Vault, and allows us to reuse wallet functionality and promote the use of the NW. In this way we can eliminate the need for a separate wallet PIN and Vault username and password.

For more advanced features in the Vault app, it will likely be necessary to have user session authentication.

As a lower priority, we can create a parallel set of functions for all Vault operations that allow a user to provide their own wallet. This could likely be done on the frontend, where a user would not need to upload a private key, but simply give access to the Vault app. Transactions would be created and signed on the client.

TODO:

* Develop an implementation plan to on-ramp fiat to Algo via Paypal/Peach and Coinbase/Luno/etc. Attention to how we ensure that we comply with restrictions for crypto exchanges, rather than simply selling our compute services?
  + Research on Paypal vs. Stripe-> Alex to test simple implementation
  + Find crypto exchange that sells us Algo in exchange for giving them fiat (or find out if Paypal sells us Algo natively) -> Alex/Co
  + Policy and jurisdiction for buying crypto with which currency and which location re. Paypals rules and etc. -> Alex + Co

# 

# Notes on Accounting System

1. **We charge a fixed percentage 5% of every (primary market) DRT sale;**
2. Because we record the current owner of each DRT within the smart contract, we can limit the execution of DRTs only to current owners, thereby enforcing that secondary market transfers are only recorded within the smart contract for a fee (which is proportional to the DRT transaction price, since we can require the reference of a transaction in which the DRT is transferred); **We then can charge a 5% fee for every secondary market transaction;**
3. **When a pool is created, we need to charge a fee proportional to the amount of initial data;** verification of a schema, per mb , times number by what we pay for storage…
4. **When someone appends data, we charge a fee proportional to the amount of data (same as with creation);** Enclave needs to tell the smart contract how much data is being added, i.e. appending 750 rows of data;
5. **When a DRT is executed, a pre-funded account will be billed upon instruction from the enclave;**
   1. Analyst sends DRT to smart contract, oracle layer checks status of blockchain, registers the DRT transaction to the smart contract and triggers code execution by the enclave;
   2. Enclave checks whether funds exist to pay for code execution [TODO: Design execution service -> Joe to do first sequence diagram];
   3. We charge in 30 minute tickets of computational time.
   4. FOR MVP: we will charge a fixed fee for the execution -

# Notes on Smart Contract

1. Data pool creators are regarded as normal “contributors” after their initial contribution and have to purchase an Append DRT in order to add additional data to the pool they created.
2. Append DRTs cannot be priced lower than 1 Algos
3. Smart contracts require a minimum balance. It is the data pool creator to fund the smart contract during its initialization.
4. Research Item: Use smart contracts box storage to keep track of pre-funded accounts (flags) or use internal accounts/wallets. Leaning towards the box storage/ flag system, but I need to research alternatives. We could make use of the local account variables to keep track?
5. [UML DIAGRAMS](#_a22q1rrfzs87)

NOTES:

1. The term “aim” means that we would like to build this functionality as part of the MVP, but if we cannot build everything, we prioritize and implement only the most important features. The term “eventually” means that it is a nice to have that is not critical for the MVP and likely to be built at a later date.
2. Later, we can facilitate functionality similar to github where code creators can set up repositories (pools) to which code contributors can add code.
3. Are data rights different from code and data? In other words, could we have a user “invent” a new data right and issue it as data or code so that others can buy DRTs to reuse it? This would mimic a license to use a DRT in a way that DRTs mimic licenses to use code and data.
4. Personal/Private Data

Multi Factor Authentication/2FA for key recovery

* + Refactor Wallet auth to use 2FA instead of multi factor authentication & Generation Agolorand account or link existing algorand account.
  + Find a way to send them algos
    - MVP: Link out to exchange.
    - Phase 2: Include exchange from Wallet (cross library) as in plan 2024

Identity Verification

1. User Wallets

There are several proposed options for handling user wallets.

The first option is to require a user to create a Nautilus wallet. This provides a level of ease of use since a user would not have to manage keys or sign transactions. They would still need to transfer funds to their wallet, so they would still be exposed to the crypto ecosystem for on-ramping. [Note: This is the option we chose because we need to be able to sign transactions on behalf of the user without requiring them to do this via external wallets]

The second option is to require a user to use their own wallet. In this case, we would either need to handle their private key, sealing it in the enclave and the Nautilus backend would handle sign transactions on a user’s behalf.

In a third case, a user would bring their own wallet, but rather than using the TEE to sign transactions, they could be signed on the client. Each user would need to give their private key to the Nautilus client app, but it would not leave the client’s computer. The Algorand Javascript SDK should be able to write and sign transactions on the user’s behalf.

1. Authentication

The issue of authentication depends on the user data that is needed by the Nautilus app and on the strategy for handling user wallets. Referring to the first and second cases in 3., the private key is protected by the enclave environment and is never exposed outside. Transactions are also restricted to those that have been generated by the relevant smart contracts. If private keys are sealed using a user password, Any API calls to the enclave would need to be accompanied by the password. An authenticated user session on the app would also be protected by the enclave sealing mechanism. Alternately, the private keys could be sealed using the enclave key and we could rely on user session authentication to protect data API calls.

TODO?: Create draft sequence diagrams for different Wallet/Authentication.

1. Data schema  
   TODO: Discuss how data schema definition looks like -> Joe to do a first pass using json schema library;

TODO: Decide whether we have a step where the data’s conformity to the specified data schema is checked; -> Json schema library can do this, Joe will check; We implement this on client side;  
TODO: What governance mechanism decides the price of DRTs? -> Co

We must enforce a data schema to:

1. Ensure interoperability of all the datasets comprising a data pool,
2. Ensure compatibility of datasets with the binary executables corresponding to DRTs.

In the case of the MVP, the most direct way to validate column data is to use a JSON standard. To ensure that different datasets can be combined and run with a given binary, each JSON dataset and each binary must specify a data schema. The JSON Schema provides a well-maintained, fast and comprehensive library in [Rust](https://crates.io/crates/jsonschema) and [Javascript](https://www.npmjs.com/package/jsonschema). It provides a media type for describing JSON documents, as well as functions to compare a JSON dataset to a JSON Schema. We have implemented a prototype that uses jsonschema to check a datafile against a schema file: [here](https://github.com/ntls-io/nautilus-trusted-compute/blob/main/rust-workspace/crates/ntc-data-packages/src/data_packages/json_schema.rs) and [here](https://github.com/ntls-io/nautilus-trusted-compute/pull/71).

Unknowns:

Where to validate? Even on an MVP and in internal testing, the rate of schema mismatch is likely to be high. We can use the Rust implementation referenced above to validate a data file in the enclave before sealing data. However, an error at this point would be slow and resource intensive. It is also not clear whether we can use the jsonschema tool in an enclave environment due to the no\_std requirement. Relying on the backend for validation then has the risk of increased development time and a clunky interface at a key point of the UI.

The alternative is to port this code to JS and check a schema on the client side, before it is encrypted and sent to our servers. This would be easier to develop and provide a better user experience.

1. Data quality  
     
   More advanced data quality validation will most likely be done after an MVP is completed.   
     
   TODO: Design selection of data rights; -> Co / Bron  
   TODO: Design workflow for sending Contributor Tokens + DRTs to the smart contract (needed to remove data and / or delete pool); -> Alex starts coding this up;
2. Removal of data by data contributors  
     
   In order for a data contributor to withdraw their data, there must not be an unredeemed execution token that references that data. The bookkeeping for the location of a datasets in a data pool, and the corresponding DRTs which reference a dataset can be tracked using Algorand boxes. However, it may not be an MVP feature. You must own the Contributor Token corresponding to the removed data. If someone buys a cheap DRT they can prevent a contributor from removing their data.

# Implementation Plan Overview

The following components are required to satisfy the above user requirements:

1. Data Management System
   1. Seal, store and validate the format of user data
   2. One or more backend APIs:
      1. Allow the frontend to transfer sealed data to storage
      2. Allow the backend to retrieve data from storage and initiate enclave execution
      3. Store relevant metadata on users, datasets,
2. Code Execution System

Handles and verifies code execution requests and delivering results

* 1. The execution enclave, implemented with Apache Teaclave and the Intel SGX SDK.

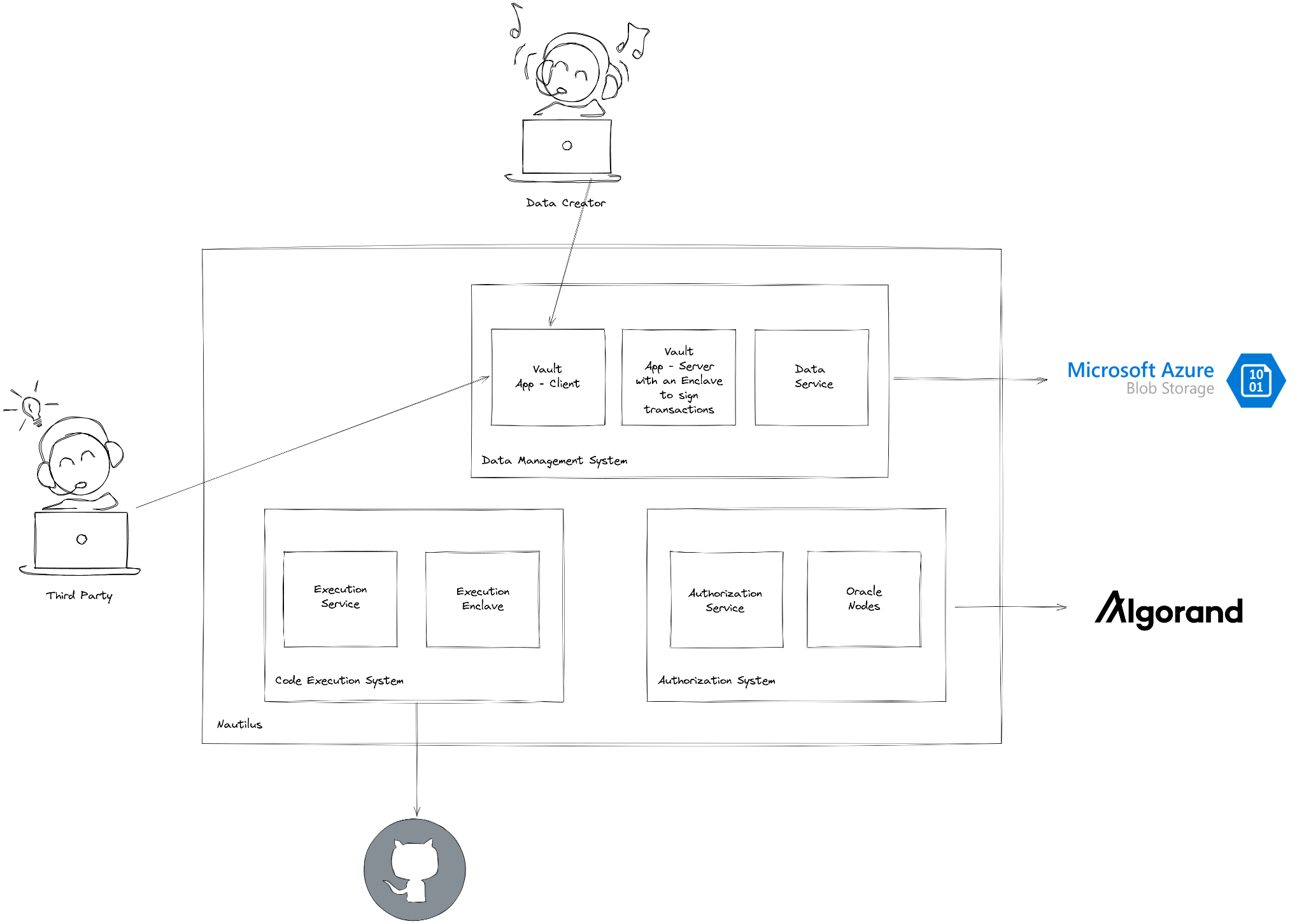
1. Authorization/Verification System

Used to verify the execution of code on data in the enclave.

* 1. Attestation using Oracle nodes
  2. Verifying data packages and compute code
  3. End-to-end encryption

We also implement smart contracts and DRT functionality

1. Digital Rights Tokens (DRTs)
   1. Smart Contracts
      1. A smart contract built using the Pyteal library deployed on the Algorand blockchain.
      2. Frontend interface written with Angular and Typescript to interact with the smart contract and with the backend



## 

## 2. 1 Authorization/Verification System

### 2.1.1 Attestation

The Authorization System is used to verify the authorization for a specific code execution request allowed by a DRT. We achieve this by calling a number of designated nodes on the distributed ledger, called *oracle nodes.*

Work required:

* The attestation is handled in simulation mode in the Nautilus Wallet. Reproduce and check the simulated validation.
* Research the requirements for accessing and using oracle nodes.

2.1.2

## 2.2 Token Management System

The token management system encompasses interactions with the blockchain such as the smart contract and the issuing of digital right tokens.

#### 2.2.1 Smart Contract Build

The first is the development of the smart contract code using the Pyteal library. This includes all the subroutines and functions contained and explained in the following [wiki](https://github.com/ntls-io/ntc-smart-contract-mvp/wiki). The output of this part are approval.teal and clear.teal files.

#### 2.2.2 Smart Contract Deployment

The automatic deployment of the smart contract to the Algorand Testnet every time a user uploads data and creates a data pool. The smart contract requires a sequence of transactions (instructions) to successfully deploy itself. The Algorand Javascript SDK is used for this deployment and the approval.teal and clear.teal files are required.

The smart contract is designed so that all the steps in creating the smart contract are successfully completed before it can start operating as a data pool.

The successful creation of a smart contract requires the following transactions in the following sequence sent to the Algorand Testnet.

1. An application creation transaction sent to the testnet blockchain to deploy the teal files, signed by the data pool creators nautilus wallet.
2. An “optin” transaction to the newly deployed application from the smart contract creator.
3. A transaction to fund the smart contract, requiring the smart contracts minimum balance.
4. An application call transaction to the smart contract from the creator to create the Contributor Token.
5. An “optin” transaction to the Contributor Token from the smart contract creator.
6. An Application call transaction to the smart contract from the creator to add the smart contract creator as a data contributor

A UML diagram of the above process can be found [here](https://drive.google.com/file/d/13E-krmtFSBKlUx4TIjJcp5oWPL3ylN_2/view?usp=share_link).

#### 2.2.3 Smart Contract API Calls

The Algorand Javascript SDK will be used to develop API calls to interact with the smart contract. UML diagrams are used to explain the method to interact with the smart contract.

API calls need to interact with the smart contract are:

* An “optin” transaction to the smart contract from an account
* An “optin” transaction to an asset from an account
* A send payment transaction to the smart contract
* Application calls from an accounts for each method
  + Create DRT method
  + Update DRT Price method
  + Add New Contributor Token method
  + Buy DRT Method in the form of a group transaction (1. App call, 2. Payment)
  + Create Contributor Token Method
  + Claim Fees Method
  + Update Data Package Method

## 2.3 Execution Management System

### 2.3.1 Vault Execution Binaries

We currently have a [basic median function](https://github.com/ntls-io/wasm-exec-sgx/tree/main/get-median-wasm) written in Rust and compiled to WASM binary. The addition of other functions is not a current priority. We also have *standard deviation* and *mean* on a branch waiting to be merged - <https://github.com/ntls-io/wasm-exec-sgx/pull/9> - Two binaries are created for each function - one for INT and one for Float.

Eventually we also want binaries for linear regression and other basic analysis and machine learning algorithms. We ultimately envision a system where a user can contribute custom code as a WASM binary and issue rights to run code.

Currently all the binaries use a byte vector - we need to investigate the use of arrays in future - this is especially important for any machine learning binaries.

Work required:

## 2.4 Data Management System

### 2.4.1 Append Function

The function to append data to an existing data pool is distinct from the other functions in that it is only used internally. For that reason it may not be necessary to execute it as a WASM binary, and instead run it as Rust code directly inside the enclave. This would have performance and maintenance advantages. The mean, median, linear regression functions, and other functions would be developed as WASM binaries for use as DRTs and as examples for external code providers.

*Currently we have two options for appending the data:*

* WASM Binary (Append Vector): <https://github.com/ntls-io/wasm-exec-sgx/pull/10>
* Append JSON (Inside enclave using Rust): <https://github.com/ntls-io/nautilus-trusted-compute/pull/70>

Append JSON can join any two JSON data sets as long as it follows the predefined format:

| {"pool": [ ]} |
| --- |

Work required:

* Update append to download the two JSON files from CosmosDB (waiting on sealing task).
  + Data downloaded in sealed format - unseal and validate schemas.
  + Update ECAL to reflect this change.
* Data validation needs to happen before joining two JSON files - need to ensure the schemas are the same.
  + It would be done when uploading, but some check should also be done on the combined dataset to insure data integrity.
* New data pool needs to be sealed and then uploaded to CosmosDB after each append task.

### 2.4.1 Metadata Database

Store data that is not available in the blockchain, HPKE, or is too slow with those methods. Use a CosmosDB deployed on Azure.

Connection details

Database schema

Collections

* Append collection
* Smart contract
* Data package
* Wallet [maybe just use the NW metadata database]
* DRT collection
* Data schema types
  + schema\_desc\_short (128 bytes) to smart contract maybe

### 2.4.2 Data Schema & Validation

Previous research on data schema definition was done by Pi on the [Data Schema Options](https://docs.google.com/document/d/1ZR25tgpHCkr3HXdLfMeo0D41GXOeFW6C_Sp3Y8xJ7co) document. An option for schema definition is the [cue dev package](https://docs.dagger.io/1010/dev-cue-package/) from Daggar. JSON validation, etc.

We currently have a custom crate for validating JSON schemas - [GitHub](https://github.com/ntls-io/nautilus-trusted-compute/tree/main/rust-workspace/crates/ntc-data-packages)

Planned user flow:

* A user initiates the data pool creation process (or the data contributor process if they are adding to an existing dataset).
* Before they upload (call an add\_data api endpoint), they should be able to search for a schema, (list\_schemas api endpoint),
  + Choose schema
* Compare new data set to selected schema
  + Ensure all fields from schema are contained in dataset
  + Check data type in each column

Work required:

* Add oracle node information to Metadata - [Line 20](https://github.com/ntls-io/nautilus-trusted-compute/blob/main/rust-workspace/crates/ntc-data-packages/src/data_packages/common.rs)
* Update crate once sealing task is completed - [Link to GitHub](https://github.com/ntls-io/nautilus-trusted-compute/blob/main/rust-workspace/crates/ntc-data-packages/src/data_packages/sealing.rs)
* add\_data api endpoint (Use the Data POST Api endpoint - [PR](https://github.com/ntls-io/nautilus-trusted-compute/pull/69))
* list\_schemas api endpoint (Can clone functionality from Data API endpoint - [PR](https://github.com/ntls-io/nautilus-trusted-compute/pull/69))

## 

### 2.4.3 Data APIs

We currently have a PR for two APIs: <https://github.com/ntls-io/nautilus-trusted-compute/pull/69>

The APIs connect to CosmosDB and can upload data and query data using specific fields. Currently this is the format for CosmosDB:

| {  "\_id" : ObjectId("63612d200d0cc3d50ecda3b5"),  "user\_id" : "Test Client 01",  "pool\_name" : "NTLS first pool",  "sealed\_data" : "This is some sealed data",  "created\_on" : {  "$date" : 1667312928083  } } |
| --- |

API endpoint: <http://127.0.0.1:8000/api/data>

**POST - upload/add data**

| curl --header "Content-Type: application/json" \  --request POST \  --data '{"id":"Test Client 01","pool\_name":"NTLS third pool","sealed\_data":"This is some sealed data"}' \  http://127.0.0.1:8000/api/data |
| --- |

**Get - View/get data**

| curl --header "Content-Type: application/json" \  --request GET \  --data '{"id":"Test Client 02"}' \  http://127.0.0.1:8000/api/data |
| --- |

Create Data Package Endpoint

* Nautilus address
* Data package hash
* Number of rows
* Smart Contract ID ( Application ID )
* Smart Contract Address
* Contributor Token ID ( Asset ID )
* DRT ID ( Asset ID )
* DRT Name
* DRT Exchange Rate
* DRT URL code binary

Work required for Data APIs:

* Set-up proper CosmosDB for NTC (CosmosDB used for testing - “ntc-data”)
* Update environment variable to include connection string to CosmosDB
* Waiting on sealing functionality - currently we are just using a placeholder String, this needs to be updated - “*This is some sealed data*”
* SSL/TLS for connecting to CosmosDB?
* API keys for specific users?

Data Storage API

* http://127.0.0.1:8000/api/data/store\_dataset
* http://127.0.0.1:8000/api/data/retreive\_dataset

## 

# 3. Technical Planning

Technical components and their requirements and time estimates for task planning. Where necessary, refer to separate technical documents for components. Fill in details for required work and implementation.

## 3.1 Data Service

In progress Create 2 APIs (upload data pool & query data pool for creator) - 1 weeks

Metadata and other functions, *See section 2.4.3* - [PR](https://github.com/ntls-io/nautilus-trusted-compute/pull/69)

In progress TEE configuration and deployment -

Set up teaclave environment - *1-2 weeks* - 70%

Migrate from sandbox to SGX hardware - *2-3 days* - 30%

Launched SGX Benchmarking

run a set of computations in SGXv1 and SGXv2 hardware to establish the performance of our technical stack. - *1-2 weeks*

In progress WASM Binary development

develop a set of basic analysis functions - mean, median, linear regression, basic clustering, etc. to use as a demonstration and an example for code providers - *2-3 weeks* - 70%

## 3.2 Execution Service

In progress Create 2 APIs (send oracle node responses to enclave & gather code and data to send to enclave) - *3 weeks*

In progress WASM runtime (research) - test with new binaries, test new hardware (link) - *3 weeks* - 70%

Testing Append Wasm binary - *1 week* - 50%

Not started Create HTTP server

Not started Validate Code Execution Request

Not started Seal result in enclave and retrieve it , Display result in file, display in interface, R2

## 3.3 Data Storage [Bingle Kruger](mailto:bingle@ntls.io)

In progress Design Cosmos Database - 2 days

In progress Create Cosmos Database - 2 days

## 3.4 Data Schema & Validation System [Bingle Kruger](mailto:bingle@ntls.io)[Joseph Bochenek](mailto:joe@ntls.io)

Not started Build a system for generating data schemas to ensure compatibility with combined data pools and to determine how code is run on data (i.e. which columns for which binaries, etc)

Not started A system to validate new files based on data schemas

In progress Search functionality and API for schema

Notes:

* JSONSchema code (link) written in the context,
* This can be done on the front end or in the enclave, probably client-side for the MVP.
* Verifying the data in very large datasets can require a lot of resources on the clients local machine.

## 3.5 Attestation [Bill Guo](mailto:bill@ntls.io)[Jean-Pierre De Villiers](mailto:jean-pierre@ntls.io)

Not started Create API to query oracle nodes, validate responses and send responses to Execution Service

Not started Change Oracle Nodes DRT structure

Not started Create SGX DCAP Attestation

Notes:

* We have mock attestation at the moment, fetch enclave report rather than the full DCAP, needed for both. Herman created a PR on the NTC repo ([SGX DCAP Attestation](https://github.com/ntls-io/sgx-dcap-attestation-rust)).
* Oracle nodes verify if DRTs are valid, check details, so they must be modified to use smart contract addresses rather than DRT tokens.

## 3.6 Authenticated Encryption

Not started HPKE Research - 1 week

Require JP input

Not started Data Sealing

Not started Data and Code Authorization

## 3.8 Smart Contracts

Testing Smart Contract Prototype build - 5 weeks

Research Smart Contract DRT append Process (Part of build but pending discussion with Co)

* Time estimate if there's a clear path - ~ 3 days includes documentation and testing
* For data to be appended we need to 1st validate the incoming data and 2nd instruct the smart contract to add the contributor so that they can receive royalties, so we can't just let anyone join unless their data has been validated.. Therefore we need some input from the enclave validation.
* Option 1 is for contributors to search for data pools to contribute towards, and after approval from the creator and validation from the enclave, the creator instructs the smart contract to add a data contributor
* Option 2 is for contributors to purchase the “Append” DRT (Could be free of charge). Upon redemption of the DRT, the users data is first validated by enclave, then if successful a transaction is sent from the redeemers wallet to add themselves as a contributor and a transaction is sent from the enclave to approve the data validation. The smart contract will only accept the new data contributor provided that there is the approval transaction from the enclave.

Not started Build the Create Smart Contract Deployment Method from SDK, Section 2.2.2 (~1 week)

Not started Smart Contract Javascript/Python SDK for APIs, Section 2.2.3 (~2 weeks)

Not started Integrate smart contract deployment with nautilus backend (~ 2 weeks)

[Smart contract requires inputs and transactions signed by enclave]

## 3.9 Vault App

Not started Create Upload Page for Vault App Frontend and link it to Data Service API

Not started Create Schema Selection Page

Not started DRT “Marketplace”

Marketplace

Not started Sign up & Register

Not started Front end of the marketplace

Not started Query the smarcontracts to retrieve DRTs

Not started Purchase process. Send algos to smartcontract

## 3.10 Documentation

In progress Revise architecture doc - 2 days - 60%

In progress Revise Implementation Plan - 3 days - 40%

In progress Revise technical documentation

## Research Items

Questions:

* Append DRT logic - meeting with Co Nov 23, 2022
* ~~Why do we need HKPE which is scheduled to consume five weeks of developer time.~~ 
  + ~~Why don’t we use the primitive sealing methods for MVP and focus on HKPE after that?~~
  + ~~Answer: we use HPKE for better encryption and metadata~~
* ~~How will the append function be created and does it need to be a binary?~~
  + ~~Why don’t we just use Rust for MVP?~~
* Restrict extra columns in the datasets or control them with schema verification - data projection or storing columns separately
* Do we need an updated SDK for memory and multi-threading
* Attestation methods - DCAP vs internal driver for EPC management

References to merge with this document:

[SystemsArchitecture\_NTC](https://docs.google.com/document/d/1veN4eiil2Lbrl23Z9OQ63Gidkd0_yw-_1zWwJ_I_ess/)

[Product Roadmap](https://docs.google.com/spreadsheets/d/1LR08J3sbHAYVz3AdkokcrFfqEoejJpS_ZX1wnumQdck/edit?pli=1#gid=225711394)

[NTC Decisions](https://docs.google.com/document/d/1sIYKeYgh8oz_ZZFQtWgPhUpEuquG-Ve7OHQAb8nL7Wk/edit#heading=h.fgm3fue14tg5)

[MVP Vault Data Processing](https://docs.google.com/document/d/1lWJMMaVluHj-5qmjkGjyzDDaUSOlbwGekgdOtfxEJUM/edit#heading=h.99dr7usurpz2)

[Trusted Compute](https://drive.google.com/drive/u/0/folders/1X5JE39hGk42ws6XtC0TQqjjtmyfrSTDg)

## [UML DIAGRAMS](https://drive.google.com/open?id=1wewzYVNUve0eXHySZvaG2PWblBAHinsr)

