# Case Study

Given that we have access already to a rendering pipeline, media cluster, marketplace and backend wallet manager, we first have to separate what will be done by the smart contracts with what will be done off chain and what interfaces are required for the users to interact with.   
  
First and foremost, since we are building a marketplace, we require a front end that is connected to our marketplace smart contract allowing trades of the assets  
  
We require a way for users to upload their projects and files to the media cluster and this will be our first smart contract, considering that the media cluster works similarly to Pinata then folders will separate the metadata and images as well as code for each project.  
Futhermore this smart contract (can also be separate) will also act as a project generator meaning it this is where the number of iterations are created and we can call this the generator contract. This will return a string pointing to the newly created folders of the user within their project.

All of this can easily be controlled via an interface so that the users never have to write their own smart contract.

In the next step, users can find an interface with various input fields allowing them to paste their ipfs link, set the number of editions (error checking required via the interface), set the price, opening time, link to their code, link to the project details which can be taken from the previous step or inserted by the user, list of splits, setting royalties and access lists. These input fields are directly correlated to variables in the smart contract that can be created upon hitting deploy and the user paying the fees.   
The smart contract itself will have predefined variables set by the user via the interface, with the possibility of editing them later. One of these variables is the whitelist location which is in the form of a link and we are using the merkle method to keep the list secure.

Some libraries we are importing are:  
 1.ERC721A since we are using fungible tokens and want to save some gas for people minting multiple nfts at once.

2.Ownable.sol to make sure that the user creating the contract is the owner and only they can change functions such as setting the price or enabling public mint.

3. ReentrancyGuard.sol to ensure that we do not have any reentrancies while minting and withdrawing at the same time.  
As an addition we can add a receive function to act as our fallback for when we are receiving ether without anyone minting

One of the tricky parts of the contract is seed generation since we do not want this to be predictable, we will avoid using blockhash or block.difficulty and instead use VRF with oracle functionality. The oracle itself in addition to the VRF can have a random number generation logic similar to our prototype oracle file we created here:

The project has a reveal and hide state which we can control by simply changing the link address which will be an onchain reveal function using the reveal function of the contract or simply filling the current link with the items we will be revealing or replacing the items of the current link with other items (note: In order to do this, we would have to avoid the use of IPFS). Lastly, we can also reveal and unreveal through the website by hiding the address and providing a placeholder container.

Our contract starts off as paused and we are using block timestamp as well as a set timer function to make sure that the minting does not start before the timer runs out.

Since we are using variables and with the correct privileges depending on the function call we are able to retrieve these variables, we are also able to send them to a database using an API while still giving access to the features of our smart contract to the users.

We will assume at this point that our wallet has access to everything and no issue signing any of these transactions but if it does have issues then we would go about implementing it in the same way we are implementing metamask features or using a different token protocols for the tokens.  
  
The last piece of our ecosystem we have to create is a contract that handles the marketplace transactions that can also act as a monitoring tool for our users to research different projects on our website.  
For the purpose of saving time we are not implementing any external libraries at the moment but normally this contract would need to be written with care as to avoid many attacks. Some of these can be simple attacks like reentrancies or fallbacks but others can be harder to spot such as delegate calls, low level attacks, poor logic or even cross site scripting similar to the attack on opensea.  
Following this, access control is a major point for all contracts and making sure the correct view types are given to functions can make or break a contract.  
Furthermore gas is another concern for all of our contracts and having a smaller time constraint would allow us to make use of calldata, assembly code and using cheaper opcode functions and using custom errors to save gas rather than asserting, reverting etc..