Information

Level of achievement	Apollo 11
Team name	SolidLah
Project name	Bazaar
GitHub	https://github.com/SolidLah/Bazaar
Link to app	https://bazaar-solidlah.vercel.app

Motivation

Through responsive web development and modern technology and tools, we hope to make the arts more accessible to everyone and create a platform for creators to express themselves and get rewarded for it. Creators can solidify legacies and make concrete impacts, imprinting their work into the everlasting blockchain, building reputation and sharing their work with the world.

Aim

We aim to create an accessible platform for:

1. Creators to mint and list their work as NFTs on the marketplace quickly and easily.
2. Buyers to find artworks of interest and participate in smooth transactions.

User Stories

- As a creator, I can mint new NFTs
- As a creator / seller, I can list NFTs for sale on the marketplace
- As a buyer, I can add NFTs on my watchlist
- As a buyer, I can showcase owned NFTs on my page
- As a buyer, I am able to follow creators who produce artwork that I admire
- As a creator / seller/ buyer, I can take part in transactions through trading NFTs in the Bazaar app
- As a creator / seller/ buyer, I want to be able to flag
- As a buyer, I am able to search for NFTs by collections and creators

Technology Stack

Frontend	
Next	**Testing**
Chakra UI	React Testing Library
Backend	
Firebase	**Testing**
	Mocha
	Chai
Blockchain	
Ethers	**Testing**
Hardhat	Mocha
Openzeppelin	Chai

Reasons for technology stack

Frontend
For the frontend, we chose Next as our web framework as it is a robust, popular, and well documented framework. It has many optimisations built in (e.g.: SSR, SSG), which can also help in the speed of our site.

For styling, we chose Chakra UI as it has a great range of base components to choose from, giving us a good foundation of building blocks to build and style our site. It also includes useful react hooks that implement many functionalities, such as toasts, overlays and popups.

Backend
As our backend, we chose Firebase as it has many features built in that facilitate our development process. Its authentication module allows us to quickly spin up email authentication, and opens up possibilities for other forms of sign up methods (google, github, …). Firestore is a noSQL database that firebase provides and combining it with the authentication module allows for easy linking of data to users.

Blockchain
As we are using the Polygon network, we will write our smart contracts in Solidity. Hardhat is a great framework for doing Solidity programming, with built in functions for compiling, deploying and testing smart contracts.

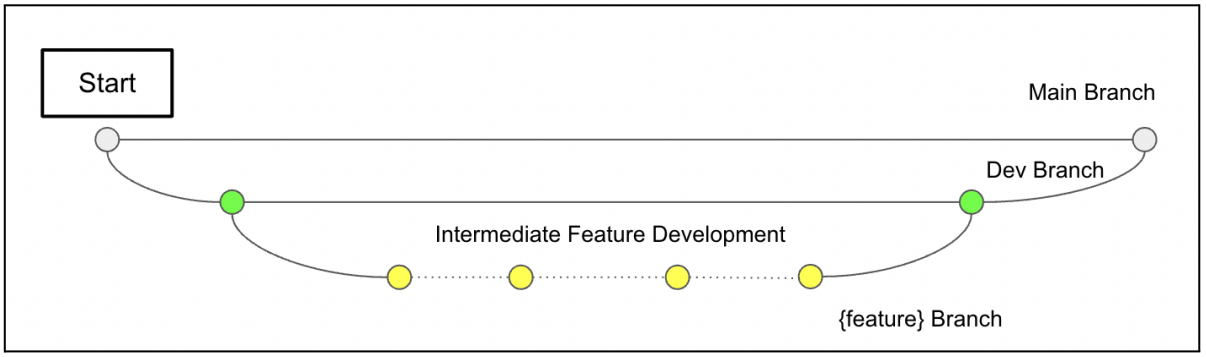
For our smart contracts, we use the contracts from Openzeppelin as a base for them, as they are widely regarded as the standard when writing smart contracts for tokens (e.g.: ERC20, ERC721). Openzeppelin contracts have security features built in (safeMint, safeTransfer, ReentrancyGuard), which protects our users from malicious actors or unintentional actions.

To interact with our smart contracts in the browser, we use Ethers library to create instances of our smart contracts in the browser, allowing us to call their functions from the browser itself.

Software Engineering Practices

Git workflow

Branching



For our git workflow, initially we have 2 branches, main and dev. Our workflow starts when we want to implement a new feature

1. Create new feature branch from dev
2. Implement the feature
3. Pull changes from dev and resolve merge conflicts
4. Create a pull request to merge the feature branch into dev
5. Test the changes and fix bugs
6. Merge the pull request into dev and delete the feature branch
7. At each milestone, dev is cleaned up and merged into main
8. A tag is created for each milestone

Issues

When we encounter a bug during testing, we file an issue under the related pull request. All bugs should be addressed and fixed before merging pull requests.

Projects

In Github, we have 3 projects, and 2 main ones, namely “Bugs and refactors” and “New features”. When creating issues for bugs, they are listed under the “Bugs and refactors” project, and Github’s projects can update the status of the issue automatically. This applies

to new features as well, where pull requests for new features are listed under the “New features” project and automatically tracked there.

Deployment

Our deployment is done through Vercel. In Vercel, the main branch is set as production, and preview deployments are created at every commit (in batches) to facilitate CI/CD.

Testing strategy

Unit testing is employed in our project to test all the components and functions in both frontend and in smart contracts. For the frontend, we will choose not to do unit testing on the screens because there is a lot of information and states handled in each screen, and it is not practical to test all of them independently. Thus, we chose to abstract out shared components such as headers, buttons and perform unit tests on those before applying them to our screens.

We also employed **manual testing** to help us identify issues with the app during actual usage and visually identify test results from the Frontend and Backend. Through this, we can better tailor the app to users by acting as users ourselves. Placing ourselves in the shoes of users will enable us to empathise with any pain points encountered and implement user-friendly changes. Any major bugs that may have slipped through unit testing can also be detected at this stage.

Note: unit testing for frontend components are not ready and will be included in Milestone 3

Frontend testing

Description	Test case	Expected	Passed
Authentication			
Email validation	Empty string	Missing fields error	T
	test	Invalid email error	T
	test@gmail	Invalid email error	T
	gmail.com	Invalid email error	T
	.com	Invalid email error	T
	@gmail	Invalid email error	T
	test@gmail.com	Successful	T
Username validation	Empty string	Missing fields error	T

	A	Successful	T
	a	Successful	T
	A test	Successful	T
	A test testing	Successful	T
Password validation	Empty String	Missing fields error	T
	String of length <= 5	Weak Password error	T
	String of length >= 6	Successful	T
Metamask wallet			
Connect a Metamask wallet	No Metamask extension	Install Metamask	T
	Have Metamask extension	Successful	T
Connect a wallet to account	No Metamask extension	Install Metamask	T
	No wallet connected	Connect wallet	T
	Wallet connected	Successful	T
Showcase owned NFTs on a “shelf”	No NFTs minted/owned	No Listings/ No owned NFTs	T
	>=1 NFT minted	Successful	T
	>=1 NFT owned	Successful	T
Mint a NFT	No Metamask extension	Install Metamask	T
	No wallet connected	Connect wallet	T
	No image uploaded	Missing fields error	T
	No name	Missing fields error	T
	No description	Missing fields error	T
	No price	Missing fields error	T
	No missing fields	Successful	T
Buy a NFT	No Metamask extension	Install Metamask	T
	No wallet connected	Connect wallet	T
	Wallet connected	Successful	T
Marketplace			
See all listed NFTs	Test if page renders	Successful	T

	successfully		
See details of individual NFTs	Test if page renders successfully	Successful	T

Smart contract testing

Description	Test case	Expected	Passed
NFT Contract			
Deployment	Name is set correctly	Successful	T
	Symbol is set correctly	Successful	T
Minting	Check balance of minter	Successful	T
	Check token URI	Successful	T
Data fetching	Check fetchNFT function	Successful	T
Metamask wallet			
Create market item function	Emit MarketItemCreated event	Successful	T
	Transfer ownership of NFT	Successful	T
	Check details of listing	Successful	T
	Price is set to zero	Reverted	T
Purchase market item function	Emit MarketItemSold event	Successful	T
	Transfer ownership of NFT	Successful	T
	Check details of transaction	Successful	T
	Invalid listing id	Reverted	T
	Insufficient MATIC	Reverted	T
	Listing already sold	Reverted	T
Data fetching	Check fetchMarketItems function	Successful	T

	Check fetchUserItems function	Successful	T

Software Engineering Principles

Abstraction

In our codebase, we wrote functions to encapsulate code often reused throughout our project such as Authentication functions as well as functional React components, keeping Abstraction and reusability of code in mind.

Incremental Development

Throughout the development process of our web application, we ensure that there is never a situation where multiple functionalities are added in a single Pull Request. Instead, as explained in our 'Branching Workflow', we branch off the dev branch for each functionality to be added, ensuring that we can test these functionalities independently in later stages, without having to worry about bugs being difficult to trace.

Single Responsibility Principle

We have designed our codebase such that every function only has responsibility over its intended functionality. For example, the functions which are called for writing to Firestore only write the corresponding data without modifying anything beyond the scope of the function's responsibility.

Anticipation of Change

In accordance with the Single Responsibility Principle and Abstraction Principle, the codebase is modularised and abstractions are written in a way to allow for changes to be easily implemented without breaking other functionalities. By doing so, we can easily tailor functionalities to fit specific needs if the need arises, without having to worry about having to rewrite the whole function.

Timeline

Milestone 1 (30 May)

Frontend	Backend
[Marketplace] Minting page	- Connect Metamask wallet - Link minting function in smart contract
[Users] Login page	- Authenticate users

[Users] Sign up page	- Add new users

Milestone 2 (27 June)

Frontend	Backend
[Marketplace] All listings view	- Fetch listings from blockchain
[Marketplace] Details view	- Fetch metadata of individual NFTs from blockchain - Business logic for purchasing NFTs (buy button)
[Marketplace] Minting page	- Feature to upload image
[Users] Profile and details view	- Name - Email - (Optional) Link a wallet to the account - If wallet is linked, display owned and listed NFTs

Milestone 3 (25 July)

Frontend	Backend
[Marketplace] Watchlist view	- Allow accounts to add and remove NFT listings in watchlist - Pull watchlist from accounts
[Marketplace] Details view (if wallet is connected to site account, show name alongside address)	- Fetch account linked to given address
[Marketplace] Details view (allow users to click on the seller’s account and view their profile)	- Protected routing to show different UI when viewing a profile externally

[Marketplace] Filtering NFTs	- Pull listings by different categories, through details that were declared during Minting
	  - Price Range
	  - Genre of Art
	  - Minter

[Marketplace] Search Bar	- Pull listings through keywords in the input field
[Users] Following creators	- Update User’s document with creator’s UID in a ‘Following’ collection

[Users] Verify email address to access user functionalities	- Enable functionality in firebase
[Users] Users can customise Profile Page with uploaded banners & Profile picture	- Update fields in Firebase with the uploaded media
Good to have	
[Marketplace] Mint in batches (collection)	- Change ethers implementation to allow for minting by collections
[Users] Additional sign up options	- Enable functionality in firebase
[Users] Use Metamask to sign login tokens for easy logins	- Interface with Metamask to send appropriate requests to sign messages

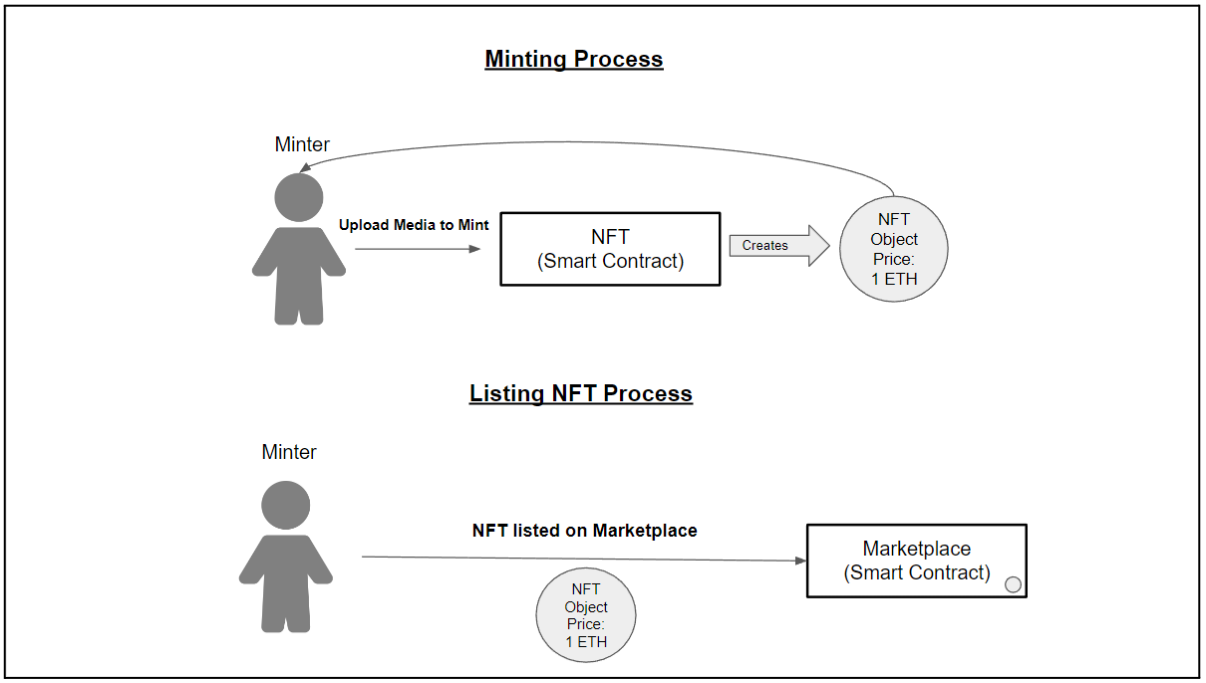
Use Cases

Description	Done
Site account	
Sign up for the site	T
Login to the site	T
Reset password	T
Connect a wallet to account	T
Showcase owned NFTs on a “shelf”	T
Add NFTs to watchlist	
Follow other creators	
View other creators profiles	
Add a custom banner on user profile	
Metamask wallet	
Connect a Metamask wallet	T
Mint a NFT	T
Mint a NFT collection	
Buy a NFT	T

Use message signing to login to site	
Marketplace	
See all listed NFTs	T
See details of individual NFTs	T
Filter NFTs by creator	
Filter NFTs by collection	

Explanation of systems

Blockchain



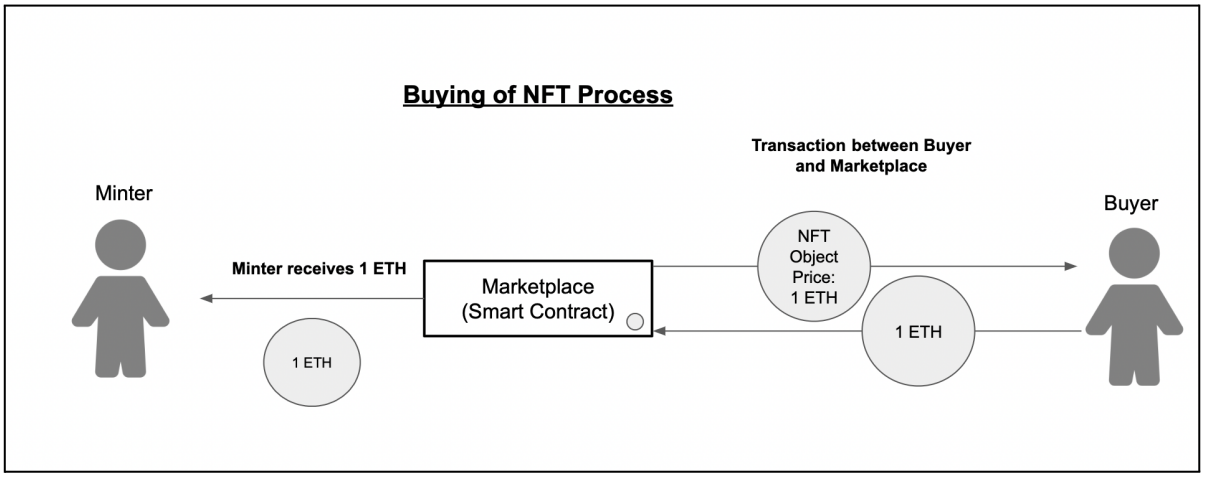
Minting Process

1. A minter uploads an image to the site
2. The image will be uploaded to IPFS and the url will be passed to the NFT smart contact on the blockchain
3. An NFT will then be minted by the smart contract with the url, tagging a unique id to each newly minted NFT
4. The NFT then sits in the minter's wallet

Listing Process

1. Once the NFT is minted, details such as sale price will be passed to the marketplace smart contract

2. The NFT is then transferred from the minter's wallet to the marketplace; the marketplace now has ownership over the NFT
3. The marketplace smart contract then creates a listing, which has; the unique id for the NFT, sales price, address of the minter, sold status of the listing, and unique id for the actual listing itself
4. The result is that every NFT listed on the marketplace will be stored on the marketplace smart contract itself, with a mapping of every NFT to the minter



Buying NFTs

1. The buyer will initiate a transaction with the marketplace smart contract through the site
2. The appropriate amount of MATIC will be withdrawn from the buyer's wallet and the sold status of that listing is updated in the marketplace smart contract
3. The withdrawn MATIC is then transferred to the minter of that NFT, as recorded in the listing
4. Once the MATIC is transferred, the NFT mapped to that listing is transferred to the buyer's wallet

Backend

API

Services

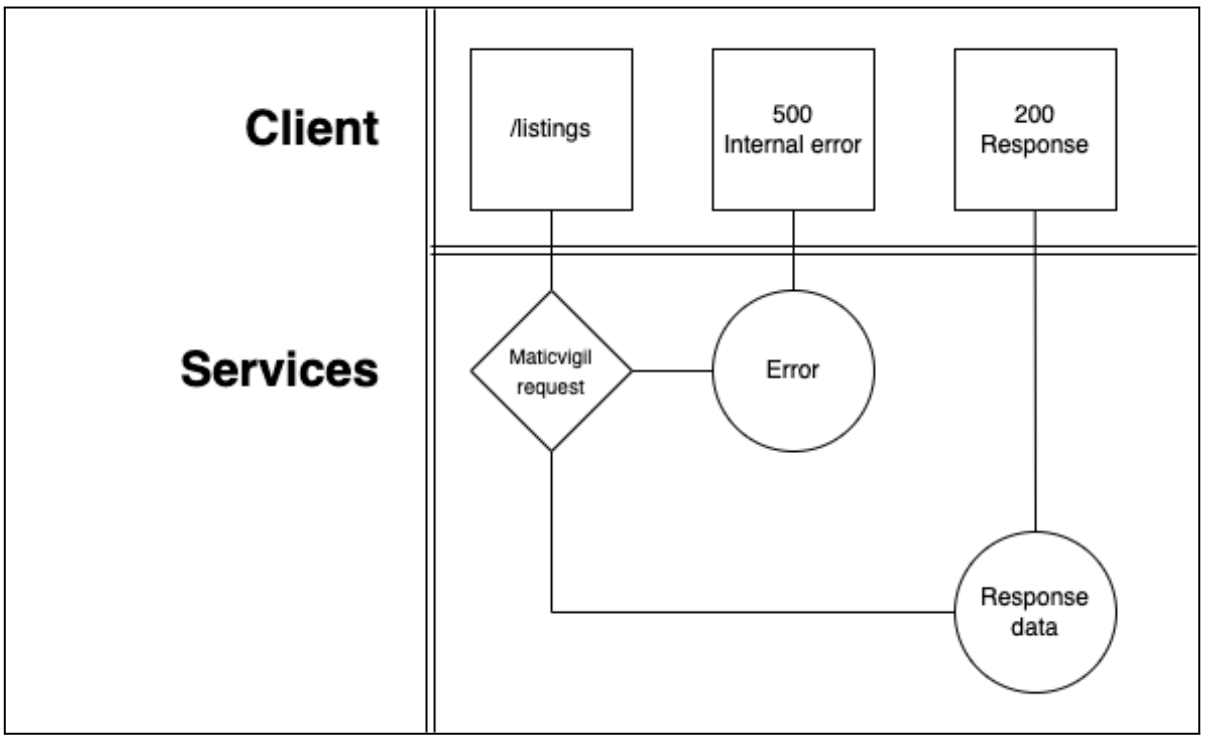
- Pinata
    - Service for pinning NFT metadata on InterPlanetary File System (IPFS)
- Maticvigil
    - Polygon network node used to send requests to the blockchain
- Ethers
    - Interface to interact with smart contracts and call their functions

Endpoints

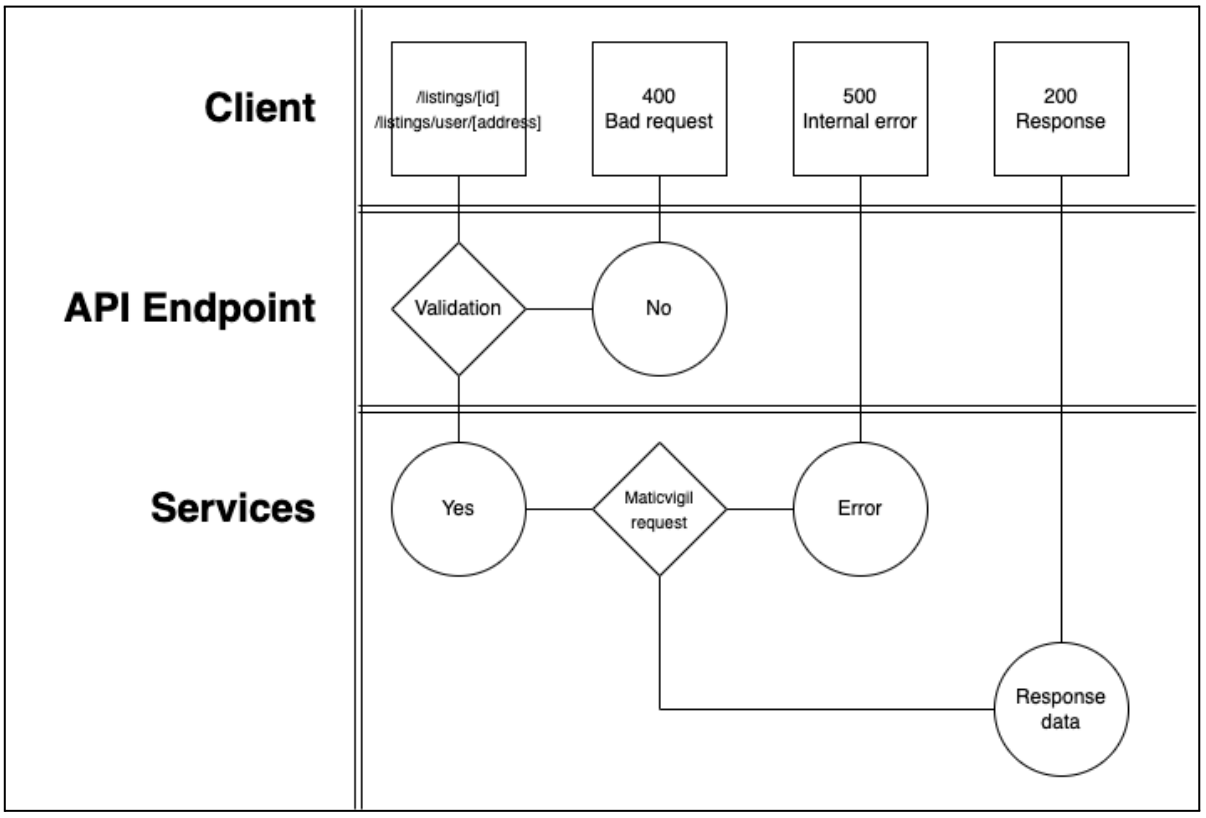
API endpoint	Description
/api/listings	Request: NIL Response: Array of all current unsold NFTs in the marketplace
/api/listings/[id]	Request: ID of market item Response: JSON object of data related to the queried ID
/api/listings/user/[address]	Request: Public wallet address of user Response: JSON object of user’s listings and user’s owned NFTs
/image	Request: Image Name Description Response: IPFS URL to metadata

Flow

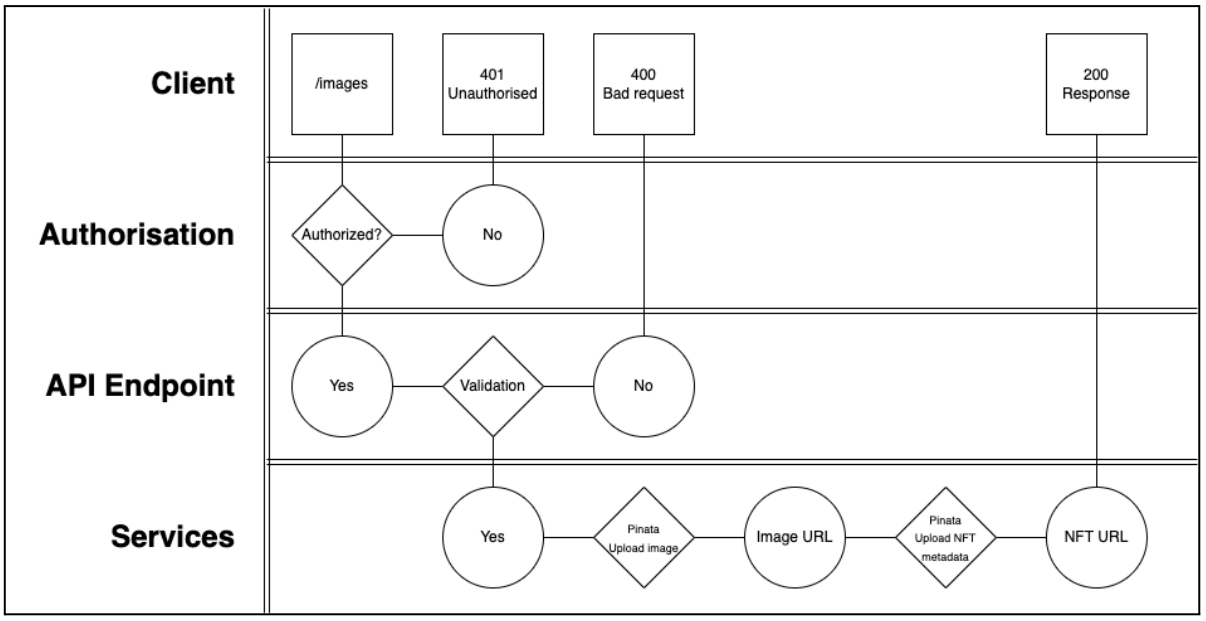
/api/listings



/api/listings/[id], /api/listings/user/[address]



/image

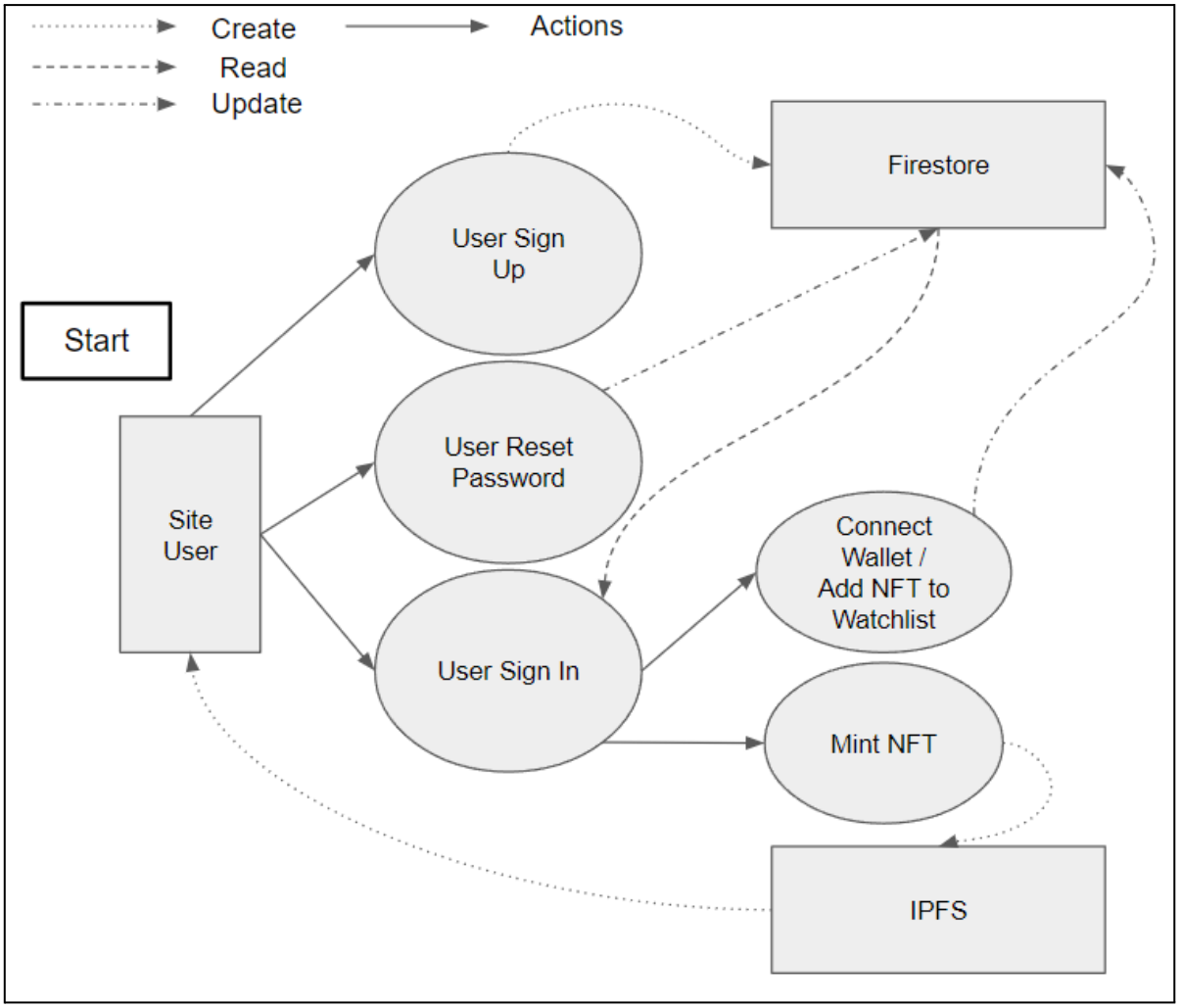


Note about the minting and listing CRUD API

During development, we realised that to create a minting and listing CRUD API, authorisation through Metamask is not possible as the browser is not involved in the API. We have decided that this is out of the scope of what we planned for our project as we planned for a webapp, and not something that can be used on the command line or through services like Postman.

Database

Flow



For our data flow design, we primarily use Firestore as our database to track user details. The first instance being when a user signs up for an account on our webapp. A new document in the ‘users’ collection is created for every user, with a unique UID being assigned to each document and storing their first name, email and wallet address . Thus, when a user wants to reset their password or sign into our webapp, data is read from their corresponding document to verify information for sign in or to update the document.

As for functions that can only be accessed after users have been signed in, users are also able to connect their Metamask wallet to their account or add an NFT from the Marketplace into their own ‘Watchlist’. We will utilise the authentication state from Firebase to update the current user’s document with the Wallet Address or TokenID into an array labelled ‘Watchlist’ respectively.

Frontend

Sitemap

URL	Description
Home	
/	Home page
Marketplace	
/marketplace	All listings page
/marketplace/details/[id]	Individual details page
/marketplace/mint	Minting page
User	
/user/me	Current logged in user’s profile page
/user/login	Login page
/user/signup	Sign up page
/user/reset	Password reset page