Group 5: AURASKY
NFT MARKETPLACE

Software Architecture Document
Version 1.4

Group ID: 05
Project Name: NFT Marketplace - AuraSky

Teacher:

Nguyen Van Vu
Ho Tuan Thanh
Tran Duy Hoang
Tran Duy Thao

Team members:

Nguyen Duc Manh - 20125012
Duong Tuan Dung - 20125025
Pham Viet Hoang - 20125031
Le Nguyen Quang Minh - 20125038

Revision History

Date	Version	Description	Author
25/Nov/23	1.0	Initial version of Software Architecture Document Manh: 2.1 Dung: 1. Hoang: 2.2 Minh: 3	Duong Tuan Dung, Nguyen Duc Manh, Pham Viet Hoang, Le Nguyen Quang Minh
29/Nov/23	1.1	Update Logical View Manh: 4.2 Dung: 4.2 Hoang: 4.1 Minh: 4.1	Duong Tuan Dung, Nguyen Duc Manh, Pham Viet Hoang, Le Nguyen Quang Minh
02/Dec/23	1.2	Update Component Diagram Hoang: 5.1, 5.2 Minh: 5.3, 5.4	Pham Viet Hoang, Le Nguyen Quang Minh
06/Dec/23	1.3	Update Deployment & Implementation View Manh, Dung: 6 Hoang, Minh: 7	Duong Tuan Dung, Nguyen Duc Manh, Pham Viet Hoang, Le Nguyen Quang Minh

Table of Contents

Software Architecture Document

1. Introduction

The AuraSky Architecture is based on a Server-client pattern with our customization to fit project requirements. The 3 following sections will give you a deep view of our architecture's design as well as the reason why we choose those approaches.

The Architectural Goals and Constraints section will show a quick view of project architecture including scope, goals, constraints as well as the team structures, development tools, and strategy when developing the software.

Use case Model represents the use-cases diagram, the most significant factor of our architecture's design.

The Logical View will give a clear and detailed description of all components as well as their functions in AuraSky architecture.

2. Architectural Goals and Constraints

○ 2.1 Architectural Goals

Technical Platform:

- AuraSky will be deployed onto our server and blockchain (our server is the application server used for internal applications and blockchain-based on the BSC testnet network).

Transaction:

- Sealze is transactional and it can leverage the technical platform capabilities. Transaction management will be reused intensively in our model platform.

Security:

- The system must be secured so that customers can make online payments safely. The application must implement basic security behaviors:
    - Authentication: connect blockchain wallets.
    - Data: double check with our server with the internal applications.
- For internet access, the following requirements are mandatory:
    - Confidentiality: sensitive data must be encrypted (blockchain payments).
    - Data integrity: Data sent across the network can not be modified by a tier.
    - Auditing: Every sensitive action can be logged.
    - Non-repudiation: gives evidence whenever a specific action occurs.

Design and implementation strategy:

- Following the RUP and SCRUM architecture to assign work for team members.
- Communicating with each other members to achieve high performance and complete all personal tasks.
- The payment process which is based on the blockchain network must not take longer than 10 seconds.

○ 2.2 Architectural Constraints

Development tools:

- Draw.io: design the component diagram

- MongoDB: database program.
- NodeJs: build backend application
- NextJs: build frontend application
- TailwindCss: CSS tools to style user interface
- Remix Eth - Truffle: deploy smart contract and design blockchain
- VSCode: developing editor
- GitHub: version control system
- Notion - Trello: project management and documents

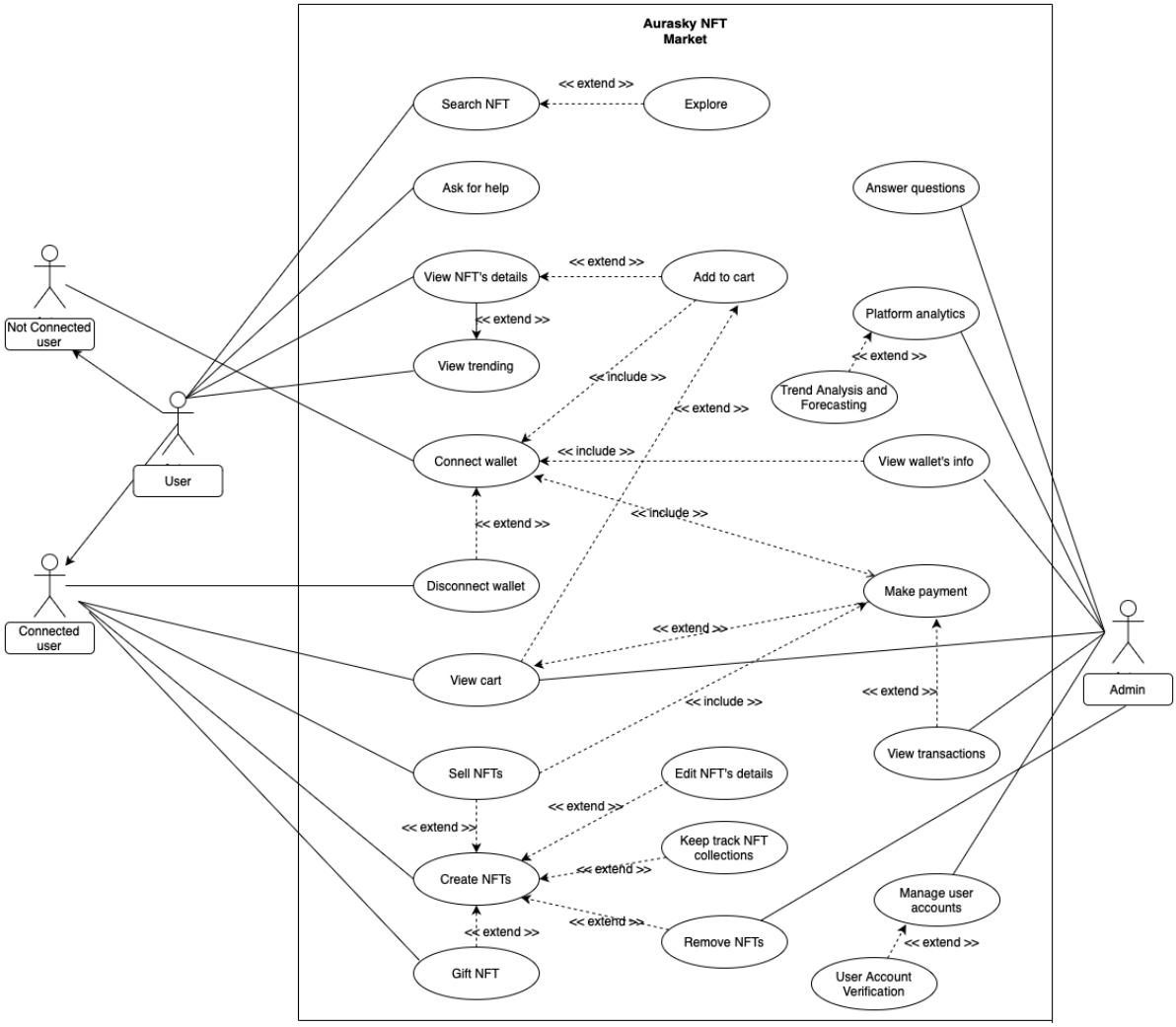
Team Structure:

- Project manager: Pham Viet Hoang
- Designer:Nguyen Duc Manh, Le Nguyen Quang Minh.
- Creative designer:Duong Tuan Dung, Le Nguyen Quang Minh
- Developer: All members.
- Tester: All members.

Schedule:

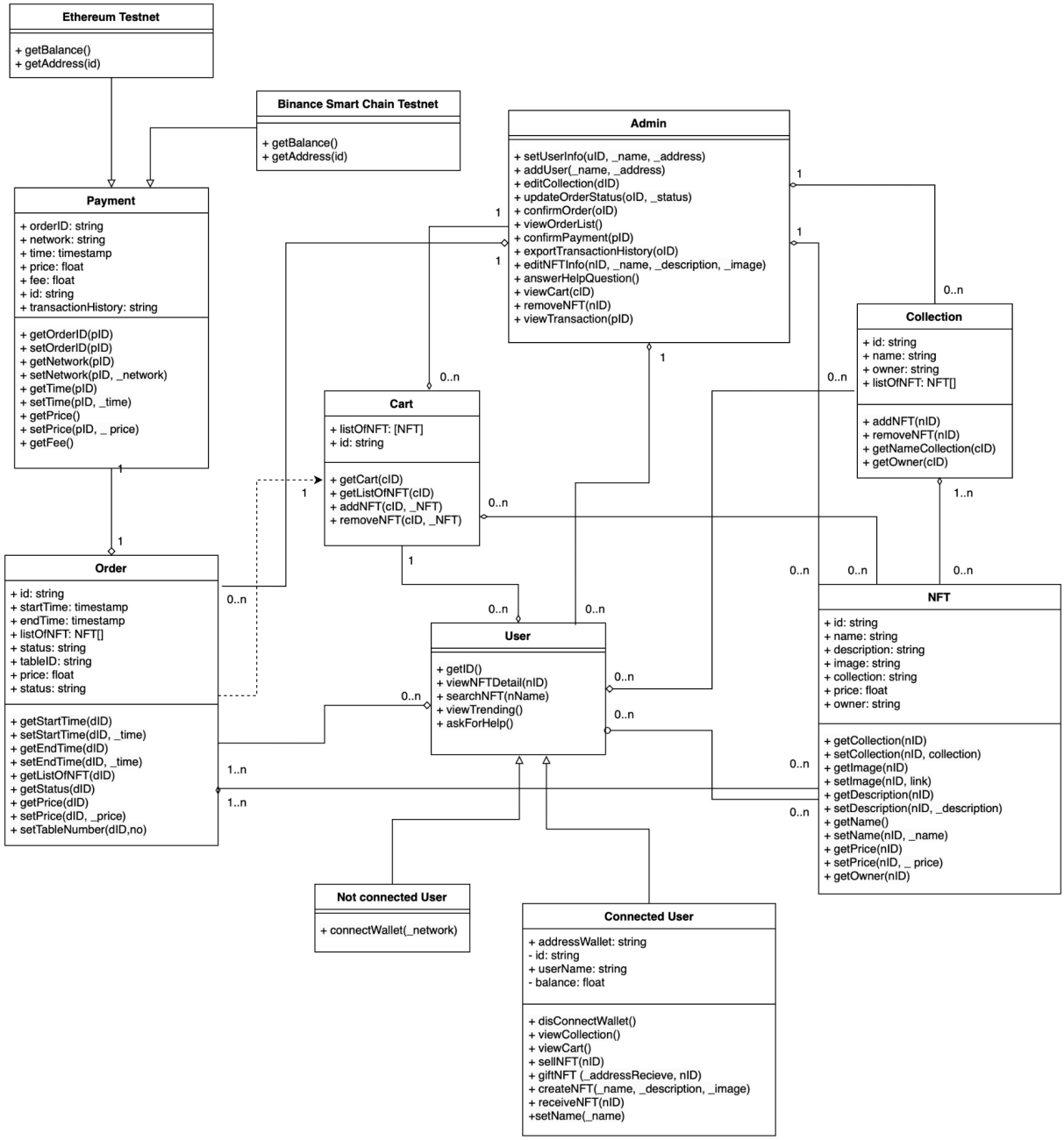
- Project Manager assigns tasks for each member of the team in the early first week.
- A print meeting will be held every Thursday, and in this meeting, each member will report what he has done, what he is working on and the issues he has.
- Furthermore, every member will work together to solve those issues.
- A planning meeting will be held every Saturday. In this meeting, the Project Manager will review all the completed work in the last week and create new work on Trello and inform other team members through Slack.

3. Use-Case Model



4. Logical View

○ 4.1 Class Diagram



Describe the detail of each class:

Abstract class Users:

Brief description: In general, the abstract User class will handle dynamic things, editing users information, and operations that all users of the web have access to such as viewing trending, NFTs details …

1. getID(): get ID account

2. viewNFTDetail(nID): View details of the name, description as well as image of the NFT
3. searchNFT(nName): Search NFT in navigation bar
4. viewTrending(): View trending and top NFT in Home Page
5. askForHelp(): Ask admin for help, you can directly inbox via Chat box

Class Not Connected User:
Brief description: The not connected user can connect to our website via the function connectedWallet()

Class Connected User:
Brief description: the Connected User class stores some basic information such as name, addressWalletAccount, id, balance …, and is able to perform the action of connected users

1. disConnectWallet(): Disconnect the current wallet
2. viewCollection(): View their own collections that storing list of NFTs
3. viewCart(): View their own cart
4. sellNFT(nID): Connected users are able to sell their NFT on the marketplace
5. giftNFT (_addressRecieve, nID): Connected users are able to send their NFT to another address wallet.
6. createNFT(_name, _description, _image): Connected users can create a new NFT, they need to provide name, description, and also upload the image.
7. receiveNFT(nID): Receive the NFT from another address.
8. setName(_name): change their display name.

Class NFT :
Brief description: the NFT class stores some basic information such as name, id, owner, description, image …, and its attributes can be retrieved and updated by setters and getters.

1. getCollection(nID): Get collection of NFT
2. setCollection(nID, collection): set collection of NFT
3. getImage(nID): Get image of NFT
4. setImage(nID, link): Set image of NFT
5. getDescription(nID): Get description of NFT
6. setDescription(nID, _description): Set image of NFT
7. getName(): Get name of NFT
8. setName(nID, _name): Set name of NFT
9. getPrice(nID): Get price of NFT
10. setPrice(nID, _ price): Set price of NFT
11. getOwner(nID): Get Owner of NFT
12. getOwner(nID, _address): Set Owner of NFT

Class Collection :
Brief description: the Collection class stores some basic information about the collection such as name, id, owner, …, and also the list of NFT

1. addNFT(nID): Add NFT to the collection
2. removeNFT(nID): Remove NFT from the collection
3. getNameCollection(cID): Get name of the collection
4. getOwner(cID): Get owner of the collection
5. setNameCollection(cID, _name): Set name of the collection
6. setOwner(cID, _addressWallet): Set owner of the collection

Class Order :
Brief description: the Order class stores some basic information about the collection such as name, id, startTime, endTime, total cost after the order is placed and confirmed by admin…, and also the list of NFT

1. getStartTime(dID): Get the time starting the order

2. setStartTime(dID, _time): Set the start time of the order
3. getEndTime(dID): Get the end time of the order
4. setEndTime(dID, _time): Set the end time of the order
5. getListOfNFT(dID): Get list of NFT in an order
6. getStatus(dID): Get status of the order
7. getPrice(dID): Get price of the order
8. setPrice(dID, _price): Set price of the order
9. setTableNumber(dID, no): Set number of table of the order

Class Admin :

Brief description: The Admin class will perform actions to manage the marketplace such as managing order status, managing tables, editing collection, view NFTs of any owner as well as adding users.

1. setUserInfo(uID, _name, _address): Edit user's information such as userID, name, address wallet
2. addUser(_name, _address): Add user to database
3. editCollection(dID): Edit NFT Collection
4. updateOrderStatus(oID, _status): Update the status of the order
5. confirmOrder(oID): Confirm the order to make a payment after
6. viewOrderList(): View list of current order
7. confirmPayment(pID)
8. exportTransactionHistory(oID): Export the transaction history of the payment
9. editNFTInfo(nID, _name, _description, _image): Edit the information of NFT such as name, …
10. answerHelpQuestion(): Answer question in Chat box
11. viewCart(cID): View cart of each users
12. removeNFT(nID): Remove the created NFT
13. viewTransaction(pID): View the transaction history of the any payment transaction

Class Cart :

Brief description: Class Cart is used to manage the NFTs that customers want to buy, including adding, removing or editing the quantity of NFTs in customer order. It also stores list of NFT and its id

1. getCart(cID): Get ID of cart
2. getListOfNFT(cID): Get list of NFT from cart
3. addNFT(cID, _NFT): Add NFT to cart
4. removeNFT(cID, _NFT): Remove NFT from cart

Abstract Class Payment :

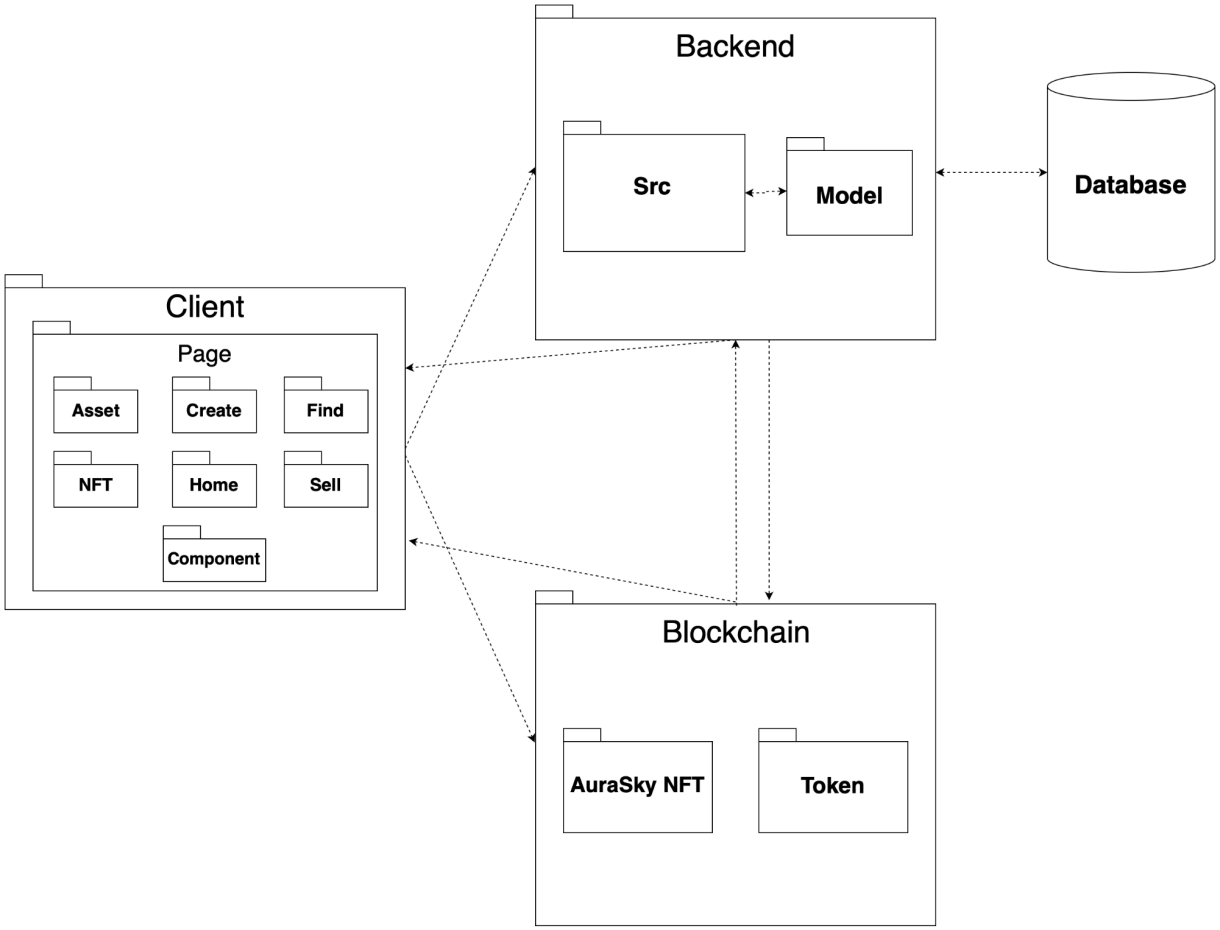
Brief description: Class Payment simply takes all the necessary user order information to make the next payment step. This class is an Abstract Class, so when a user uses any network to make a payment, the following legacy layers will process that payment through the methods. It contains a lot of attributes such as orderID: string, network, time, price, fee, id...

1. setOrderID(pID, _id): Set order ID
2. getNetwork(pID): Get current network to make a payment
3. setNetwork(pID, _network): Set network of wallet to make a payment
4. getTime(pID): Get time to make a payment
5. setTime(pID, _time): Set time to make a payment
6. getPrice(): Get a total price need to pay
7. setPrice(pID, _ price): Set a total price need to pay
8. getFee(): Get fee
9. getOrderID(pID): Get order ID

Class Ethereum Testnet and Binance Smart Chain Testnet :

Brief description: Class Ethereum and Binance Smart Chain testnet inherit from Abstract Class Payment is used to make the payment in Ethereum network or Binance Smart Chain Testnet

○ 4.2 Package Diagram



Describe the detail of each package:

Client Package :

Brief description: Client Package contains a Page package including:

- 7 folders: asset, create, find, NFT, home, sell, component
- 3 files_app.tsx, _document.tsx, index.tsx

Moreover, 7 folders have its functionality. I will explain below:

- **create folder** stores index.tsx, which plays a role to create NFT, then call API and update to Backend)
- **asset folder** stores index.tsx, which plays a role to render UI asset pages, first call API to get NFT lists and render on UI.
- **find folder** stores index.tsx, which plays a role in function searching NFT on navigation bar, then call API to find in database
- **NFT folder** stores index.tsx, which plays a role to render NFT card

- **home folder** stores index.tsx, which render UI in homePage
- **sell folder** stores index.tsx, which plays a role to render sell page, then call API and update to Backend
- **component folder** stores navigationBar.tsx, connectWallet.tsx, web3.tsx, contractNFT.tsx ... many components can be reused will be stored on this folder

Blockchain Package :

Brief description: Blockchain Package contains a 2 packages including:

- AuraSky NFT: contains source code including AuraSkyNFT.sol contract writing in Solidity, deploy files in Javascript, migration files in Javascript …
- Token: contains source code including Token.sol contract writing in Solidity, deploy files in Javascript, migration files in Javascript …

Backend Package :

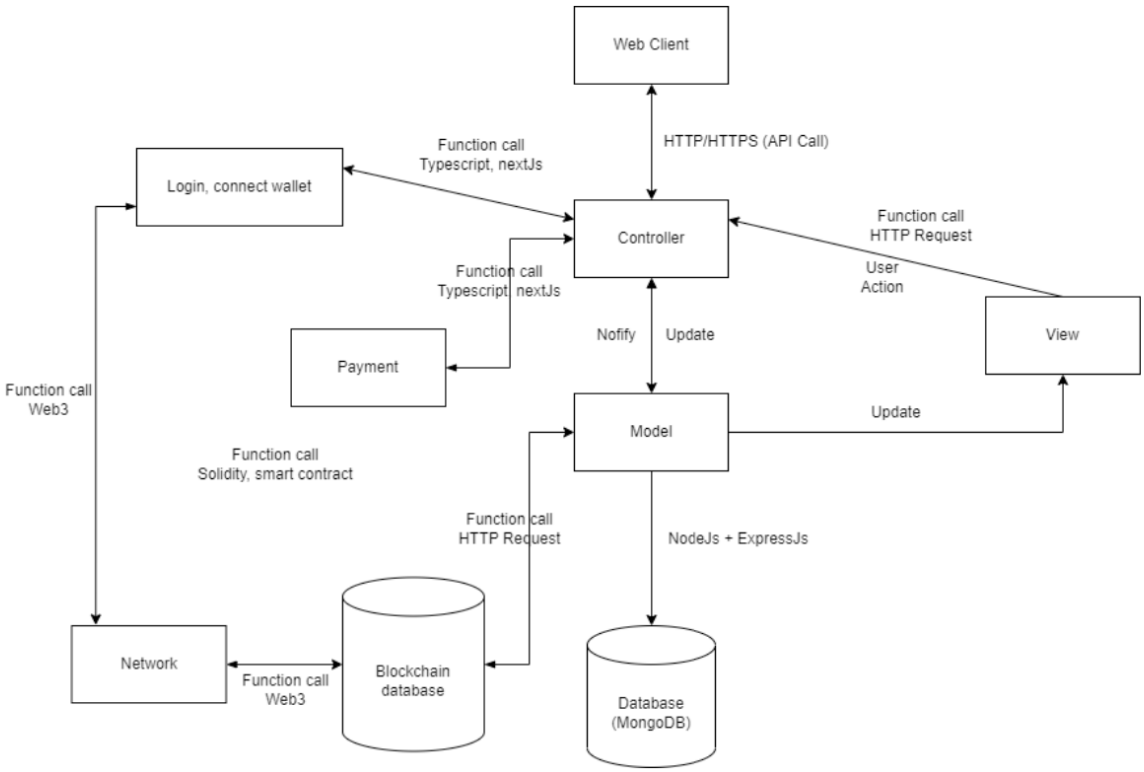
Brief description: Backend Package contains Src and Model package including:

- Src: Routes folder, Server.ts and Utils
  - idMintedRoute.ts: Route for minting NFT
  - moneyRoute.ts: Route for claiming money and sent money to contract
  - nftCreateRoute.ts: Route for creating NFT
  - nftRoute.ts: Route for NFT attributes
  - userRoute.ts: Route for user attributes

  and
  - config.ts
  - log.ts
- Model:
  - idMintedModel.ts: Model for minting NFT
  - moneyModel.ts: Model for claiming money and sent money to contract
  - nftCreateModel.ts: Model for creating NFT
  - nftModel.ts: Model for NFT attributes
  - userModel.ts: Model for user attributes

-

5. Component



Create: AuraSky allows connected users to create unlimited NFTs with zero fees until users sell them on the marketplace.

Payment:

- **Sell:** Whenever users want to sell an NFT created by him/her on AuraSky, they have to pay for our contract as a fee to mint this NFT from blockchain and add value to the NFT they want to sell. ⇒ Therefore, AuraSky needs a pseudo database to store these temperature NFTs until they are minted and added to the blockchain. When NFTs are sold, the user can get money.
- **Buy:** The connected user can buy any NFTs on the marketplace. They have to pay the fee for the blockchain and the amount of value of this NFT to keep them in their wallet.

View and search: based on the users' key.

Database: Our database is the application server used for internal applications that store the NFTs' properties and keep track of the transactions.

Blockchain: Any actions such as selling, buying, and connecting wallets need to go through blockchain. Blockchain double- checks with our database to make the transaction.

○ 5.1 Component: Server

According to the above description, data is stored in 2 places: blockchain and pseudo database. Therefore, we have 2 problems:

- Data from different sources may be in different formats ⇒ hard for the client component to handle.
- Security problem: hackers can read the database URL easily on web browsers if we store it on the

frontend.

⇒ To solve both problems, AuraSky uses a pseudo server.

The server acts as a pseudo layer between the blockchain and the client.

Obtains and manages data from the database.

Normalizes the data obtained from the database and then sends them to the client.

The server is implemented in NodeJs language and ExpressJs framework. The AuraSky server is deployed on the Vercel platform.

○ 5.2 Component: Blockchain

- Blockchain, in fact, is the server component in the Server-client pattern. Blockchain handles all transactions from the users' trading activities and stores data of users' assets on the marketplace.
- Blockchain, or smart contract, is implemented in Solidity language and deployed using Truffle.

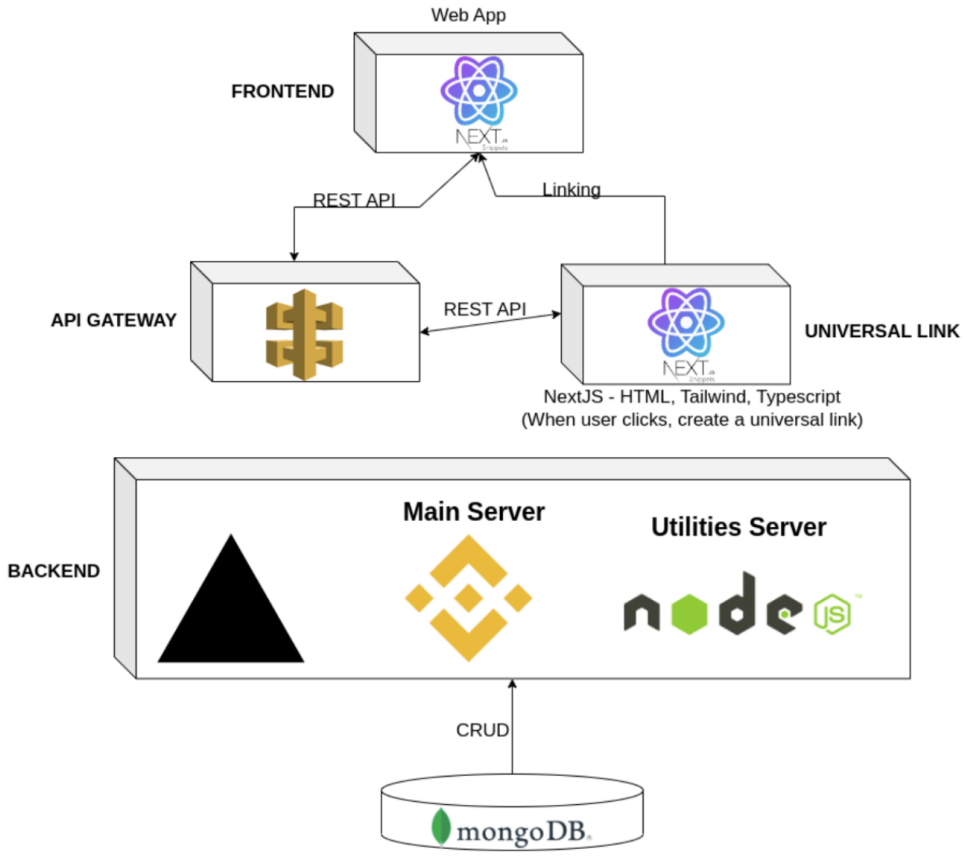
○ 5.3 Component: Pseudo Database

- The pseudo database stores:
  - Temperature NFTs data before they are minted and added to the blockchain.
  - Users' settings and personal information on the AuraSky marketplace.
- In order to publish the marketplace to users as soon as possible, we decided to use a NoSQL database, MongoDB. Additionally, the pseudo database is connected to the pseudo server using mongoose.

○ 5.4 Component: Client

- Client is the frontend application that manages the user interface and user experience (UI/UX).
- The client specifies resources to be fetched from the pseudo server corresponding to users' interaction.
- Besides that, in order to connect users to the blockchain, clients have to handle Metamask Wallet connection.
- The frontend of Marketplace is implemented in NextJs and styled by Tailwind CSS. Metamask Wallet connection is handled by Etherium Provider API.

- ## 6. Deployment



- ## 7. Implementation View

BackEnd: https://github.com/hoangrank1/AuraSky_NFT-Market/tree/main/src/Backend

- build/
  - models/
    - nftModel.js
    - userModel.js
  - routes/
    - nftRoute.js
    - userRoute.js
  - utils/
    - config.js
    - log.js
  - server.js
- src/
  - models/
    - idMintedModel.ts
    - nftModel.ts
    - userModel.ts
  - routes/
    - idMintedRoute.ts
    - nftRoute.ts
    - userRoute.ts

    - utils/
        - config.ts
        - log.ts
    - server.ts
- package-lock.json
- package.json
- postman.json
- tsconfig.json
- .env

IPFSService: https://console.firebase.google.com/u/2/project/cs300-nftmarketplace/overview

- src/
    - config.ts
    - index.ts
    - ipfs.ts
- .env
- package-lock.json
- package.json
- tsconfig.json

Blockchain: https://github.com/hoangrank1/AuraSky_NFT-Market/tree/main/src/Blockchain

- AuraSky NFT/
    - build/
        - contracts/
            - Address.json
            - Context.json
            - ERC165.json
            - ERC721.json
            - IERC165.json
            - IERC721.json
            - IERC721Metadata.json
            - IERC721Receiver.json
            - NFTMinh.json
            - Ownable.json
            - AuraSkyNFT.json
            - Strings.json
    - contracts/
        - AuraSkyNFT.sol
    - migrations/
        - 1_initial_deploy.js
    - package-lock.json
    - package.json
    - secrets.json
    - truffle-config.json
- Token test/
    - build/
        - contracts/
            - Context.json
            - IBEP20.json
            - Minh.json

        - Ownable.json
        - SafeMath.json
    - contracts/
        - Minh.sol
    - migrations/
        - 1_initial_deploy.js
    - package-lock.json
    - package.json
    - secrets.json
    - truffle-config.json

FrontEnd: https://github.com/hoangrank1/AuraSky_NFT-Market/tree/main/src/Frontend

- components/
    - card_nfts_1.tsx
    - card_nfts_2.tsx
    - navigationbar.tsx
    - categorybar.tsx
- pages/
    - home.tsx
    - nft.tsx
    - connect.tsx
    - explore.tsx
    - _app.tsx
    - creat.tsx
    - index.tsx
    - account.tsx
    - cart.tsx
- public/
    - webIcon .ico
    - vercel.svg
- styles/
    - globals.css
- utils/
    - config.ts
- next.config.js
- package.json
- postcss.config.js
- tailwind.config.js
- tsconfig.json