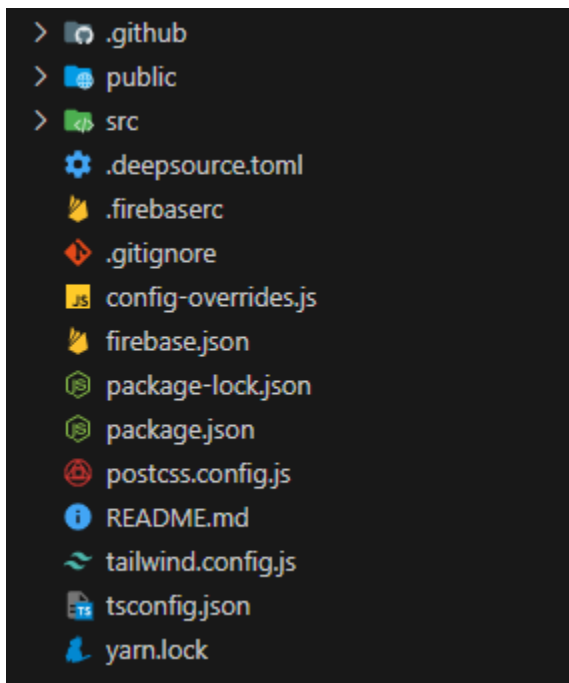


คู่มือสำหรับนักพัฒนา

เทคโนโลยีที่โดยรวมของระบบ NFT-Marketplace จะมีการพัฒนาอยู่ 3 ส่วนดังนี้ frontend, backend, smart contract

1. Frontend

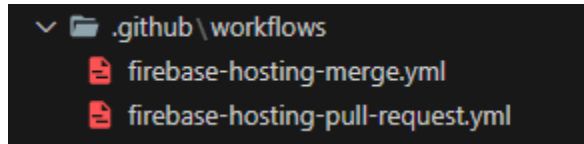
frontend โดยระบบของ NFT-Marketplace ใช้ library React , Bootstrap สำหรับการทำให้ UI และทำการใช้ Redux เพื่อจัดการ state ในระบบ NFT-Marketplace โดยจะเห็นได้ว่าในระบบจะมีโครงสร้างทั้งหมดดังนี้



รูปที่ 1 frontend directory

ในภาพไฟล์ package สำหรับ project, ไฟล์ config ของ firebase สำหรับการเรียกใช้การสำหรับการ deploy ,ไฟล์ config สำหรับ tailwind css และจะมี folder อยู่ทั้งหมด 3 folder ดังนี้

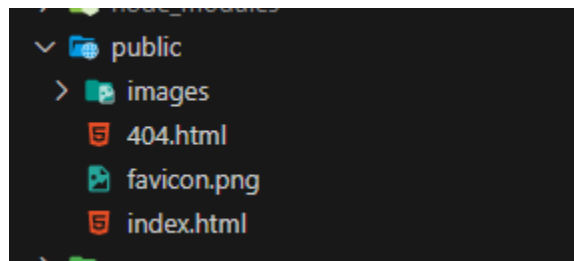
1) github folder



รูปที่ 2 workflows สำหรับ deploy

ซึ่งทั้ง 2 ไฟล์นี้เป็นไฟล์สำหรับการทำ workflow สำหรับการทำ auto deploy สำหรับตัว project ใน github

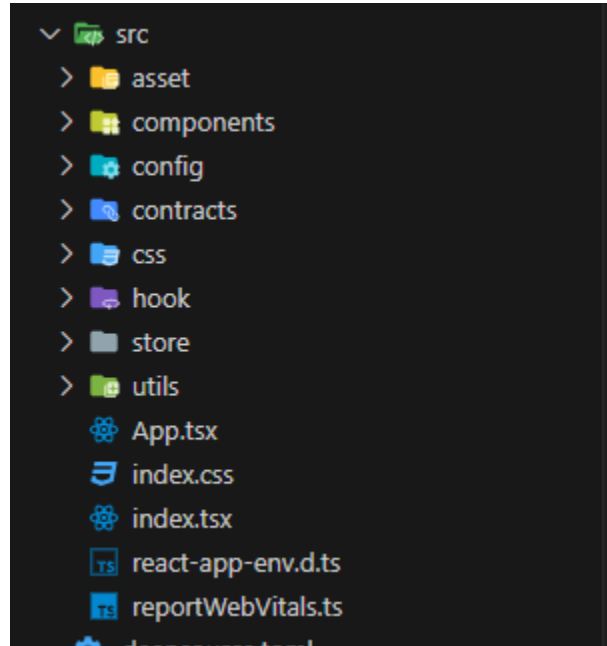
2) public folder



รูปที่ 3 public folder สำหรับแสดง interface

ใน folder public จะไฟล์ index.html ที่ใช้สำหรับการเรียกใช้ไฟล์ root ของ react เพื่อแสดง UI และมีไฟล์ favicon.png เป็น logo ใน header ของเว็บ และไฟล์ 404.html แสดงผลการไม่เจอหน้า UI และใน folder images คือ folder ที่เอาไว้เก็บรูปภาพที่เกี่ยวข้องกับตัวหน้าเว็บ

3) src folder



รูปที่ 4 src directory

จากภาพเป็นโครงสร้างใน folder ของ src โดยที่มามีการทำงานจากการ render ใน file index.tsx ในไฟล์ นี้จะเป็นการ render root ที่มี component App ภายใน provider ของ redux store และมีการ import bootstrap css framework ที่ใช้สำหรับการ styling UI

1.1 Store ของระบบ

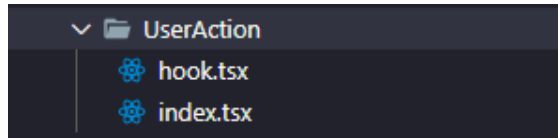
```
1 import { configureStore } from "@reduxjs/toolkit";
2 import { useDispatch } from "react-redux";
3 import { save, load } from "redux-localstorage-simple";
4
5 import userAccount from "./UserAction";
6 import transactionRes from "./TransactionAction";
7 import jwt from "./JWTAction";
8
9 const PERSISTED_KEYS = {
10   states: ["userAccount", "jwt"],
11   namespace: "app",
12 };
13
14 const store = configureStore({
15   reducer: {
16     userAccount: userAccount,
17     transactionRes: transactionRes,
18     jwt: jwt
19   },
20   middleware: (getDefaultMiddleware) =>
21     getDefaultMiddleware({ thunk: true }).concat(save(PERSISTED_KEYS)),
22   preloadedState: load(PERSISTED_KEYS),
23   devTools: process.env.NODE_ENV !== "production",
24 });
25
26 export type AppDispatch = typeof store.dispatch;
27 export type AppState = ReturnType<typeof store.getState>;
28 export const useAppDispatch = (): AppDispatch => useDispatch();
29
30 export default store;
```

รูปที่ 5 store ของระบบ

ในไฟล์ store/index.tsx นี้จะมีการ import slice ที่ใช้ใน ระบบ ดังนี้ userAccount, transactionRes, jwt โดยมีการกำหนดชื่อของ state สำหรับการนำข้อมูลไปเก็บไว้ใน localStorage และทำการสร้าง configureStore ในชื่อ store โดยนำ slice ที่ import มาสร้าง reducer และกำหนด middleware โดยนำ state ที่เราสร้างมา save ข้อมูลเก็บไว้ใน localStorage

Reducer ทั้ง 3 ที่ได้ import มีดังนี้

- 1) UserAction
- 2) TransactionAction
- 3) JWTAction



รูปที่ 6 โครงสร้างของ reducer

ใน UserAction มีไฟล์อยู่ 2 ไฟล์ คือ index.tsx, hook.tsx โดยที่ไฟล์ hook มีไว้สำหรับการเรียกใช้งานข้อมูลหรือ function สำหรับการเขียนข้อมูลใน state สำหรับ Object ส่วนไฟล์ index.tsx จะเป็นการจัดการข้อมูลใน state ของ store และ folder store มีไฟล์ type.ts เพื่อใช้ export interface ดังนี้

```
1  export interface userAccountProps {
2      address: any;
3      profileImg: string;
4  }
5
6  export interface TransactionResProps {
7      waitTransactionState: Boolean;
8  }
9
10 export interface JWTProps {
11     refreshJWT: string;
12 }
13
```

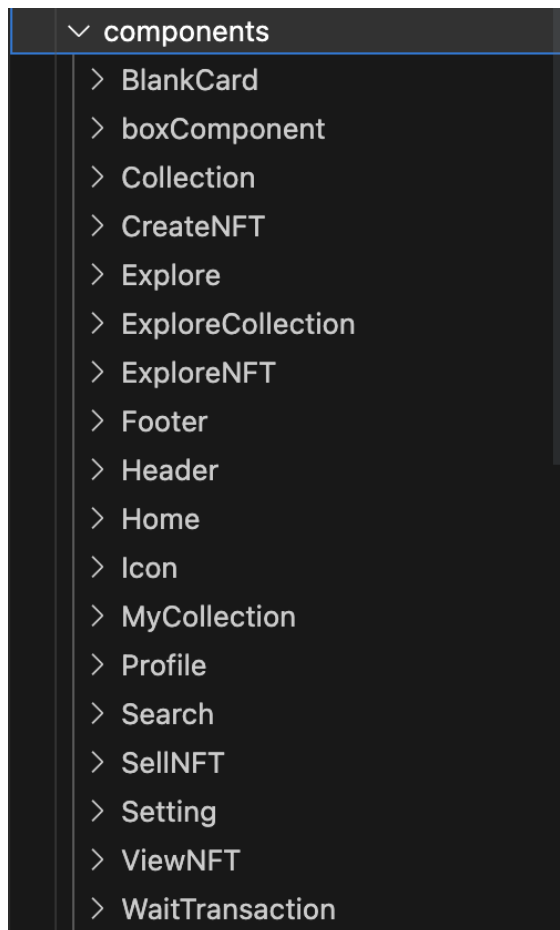
รูปที่ 7 interface for store

1.2 Component App

```
16 function App() {
17   return (
18     <div>
19       <BrowserRouter>
20         <Header />
21         <Routes>
22           <Route path="/" element={<Home />}></Route>
23           <Route path="/profile/walletAddress" element={<Profile />}></Route>
24           <Route path="/setting" element={<SettingPage />}></Route>
25           <Route path="/createNFT" element={<CreateNFT />}></Route>
26           <Route path="/myCollection" element={<MyCollection />}></Route>
27           <Route
28             path="/collection/:collectionId"
29             element={<Collection />}
30           ></Route>
31           <Route path="/viewNFT/:tokenId" element={<ViewNFT />}></Route>
32           <Route path="/sellNFT/:tokenId" element={<SellNFT />}></Route>
33           <Route path="/search/:searchValue" element={<Search />}></Route>
34           <Route
35             path="/viewNFT/"
36             element={<ExploreNFT isSaleList={false}></ExploreNFT>}
37           ></Route>
38           <Route
39             path="/viewSaleNFT/"
40             element={<ExploreNFT isSaleList={true}></ExploreNFT>}
41           ></Route>
42           <Route
43             path="/collection/"
44             element={<ExploreCollection></ExploreCollection>}
45           ></Route>
46         </Routes>
47       </BrowserRouter>
48     </div>
49   );
```

รูปที่ 8 App.tsx สำหรับการจัดวาง frontend

สำหรับ app.tsx ในโฟลเดอร์ src นั้นจะเป็นส่วนเริ่มต้นของ react โดยมีการสร้าง tag Header เพื่อใช้ในทุก route ซึ่งจะทำหน้าที่ path ไปยัง component อื่นซึ่งจะประกอบด้วย component ดังต่อไปนี้



รูปที่ 9 component structure

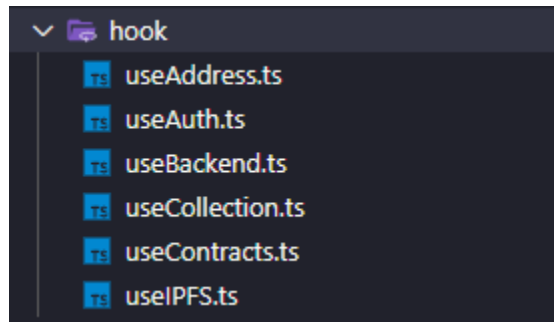
โดยในโครงสร้างของ component structure แต่ละ component จะมีหน้าที่ดังต่อไปนี้

- 1) BlankCard มีหน้าที่ในการสร้าง BlankCard เพื่อรองรับการ fetch ข้อมูล
- 2) boxComponent มีหน้าที่สำหรับการสร้าง Card สำหรับแสดงข้อมูล NFT และ Collection
- 3) CreateNFT มีหน้าที่ในการ render หน้า createNFT สำหรับการสร้าง NFT
- 4) Explore เป็น Component เพื่อใช้ในการปรับแต่งของหน้า ExploreCollection และ ExploreNFT
- 5) ExploreCollection มีหน้าที่ในการ render เพื่อแสดงข้อมูล Collection
- 6) ExploreNFT มีหน้าที่ในการ render เพื่อแสดงข้อมูล NFT
- 7) Footer มีหน้าที่ในการ render ส่วนของ Footer ให้มีพื้นที่ว่างที่เหมาะสม
- 8) Header มีหน้าที่ในการ render Navbar สำหรับการ Search และการ path
- 9) Home มีหน้าที่ในการ render หน้าหลักของ Application

- 10) MyCollection มีหน้าที่ในการ render หน้าของผู้ใช้งานสำหรับ MyCollection
- 11) Profile มีหน้าที่ในการ render ส่วนของหน้าผู้ใช้งาน
- 12) Search มีหน้าที่ในการ render และจัดการการรับ parameters ในส่วนของการ Search
- 13) SellNFT มีหน้าที่ในการ render หน้าสำหรับการตั้งราคาขาย
- 14) Setting มีหน้าที่ในการ render หน้าสำหรับการตั้งค่าเปลี่ยนโปรไฟล์ของ User
- 15) ViewNFT มีหน้าที่ในการ render หน้าสำหรับการแสดง NFT
- 16) WaitTransaction มีหน้าที่ในการ render ส่วนของ Modal Popup สำหรับรอการยืนยัน Transaction

โดยในแต่ละ component นั้นจะมีโครงสร้างหลัก ๆ ประกอบด้วย index.tsx ทำหน้าที่จัดการในส่วน of HTML และ TypeScript โดยที่ frontend จะมี hook ในการทำ action หรือเรียกใช้งาน services ต่าง ๆ นอกเหนือจากนี้ในบาง component จะมี style เป็นของตัวเองจึงได้มี .css เพื่อใช้เก็บ style สำหรับ component นั้นๆ

1.3 Hook and services



รูปที่ 10 hook structure

สำหรับ Hook and Service นั้นจะเป็นส่วนที่ทำหน้าที่ในการทำ logic ต่าง ๆ ที่เกี่ยวข้องกับ การติดต่อไปยัง backend, IPFS และส่วนของ smart contract ซึ่งจะแบ่งประเภทออกมาดังต่อไปนี้

- 1) useAddress.ts มีหน้าที่จัดการ service และ state ที่เกี่ยวข้องกับ address ของ wallet
- 2) useAuth.ts มีหน้าที่จัดการ service และ state ที่เกี่ยวข้องกับการ Login, Logout
- 3) useBackend.ts มีหน้าที่จัดการ service และ state ที่เกี่ยวข้องกับการติดต่อกับส่วนของ backend ผ่านการใช้ axios ซึ่งเป็น library ที่ถูก Import มาใช้งานในการใช้ HTTP request
- 4) useCollection.ts มีหน้าที่จัดการ service และ state ที่เกี่ยวข้องกับการ Collection ผ่านการ fetch data จาก backend

- 5) useContracts.ts มีหน้าที่จัดการ service และ state ที่เกี่ยวข้องกับ Smart Contract โดยมีการเรียกใช้งาน library จาก Web3 และ Ethers เพื่อเข้าถึง Blockchain
- 6) useIPFS.ts มีหน้าที่จัดการ service ที่เกี่ยวข้องกับการใช้ IPFS

1.4 Config

ในส่วนของ Config นั้นจะเป็นส่วนของการตั้งค่าใน frontend เป็นในลักษณะ global เพื่อให้ง่ายต่อการแก้ไข variables ที่ถูกเรียกใช้บ่อยสำหรับการตั้งค่าต่าง ๆ ภายในระบบ โดยจะมี 3 ส่วนคือ

- 1) abi.json สำหรับ interface CONTRACT_ADDRESS เพื่อให้ง่ายต่อการเรียกใช้ function ใน smart contract ส่วนของ NFT
- 2) abi2.json สำหรับ interface Market_ADDRESS เพื่อให้ง่ายต่อการเรียกใช้ function ใน smart contract ส่วนของ Marketplace
- 3) index.js สำหรับการ exports variables ที่นำไปใช้ในการตั้งค่า application

```
1 module.exports = {  
2   baseUrl: process.env.REACT_APP_BACKEND_API_SERVICE,  
3   CONTRACT_ADDRESS: "0x985F253fB2F1b47acAAA6fcdc1D00178f7E7B207",  
4   Market_ADDRESS: "0xE810a1Fa602E71548e3E40E0Db0109fF47A9B7D9",  
5   blockchainName: "Sepolia",  
6   chainID: "0xaa36a7",  
7   jwtSecretKey: process.env.REACT_APP_JWT_SECRET_KEY,  
8 };  
9
```

รูปที่ 11 Config สำหรับ frontend

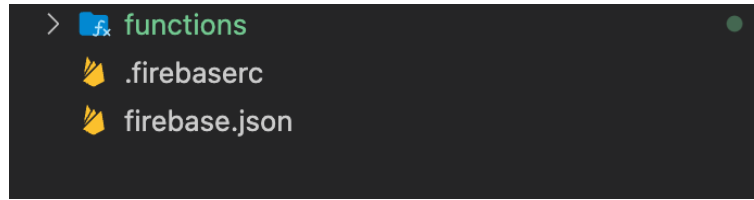
1.5 Package version

ในส่วนของ version ที่ใช้ในการพัฒนาและ support นั้นสำหรับ dependencies สามารถดูได้จาก package.json โดยจะมี package หลักที่สำคัญประกอบด้วย package ดังต่อไปนี้

- 1) react/react-dom version 18.1.0
- 2) npm version 5.1.0
- 3) axios version 0.27.2
- 4) ethers version 5.7.0
- 5) web3 version 1.7.5
- 6) bootstrap version 5.2.0
- 7) @reduxjs/toolkit 1.8.3

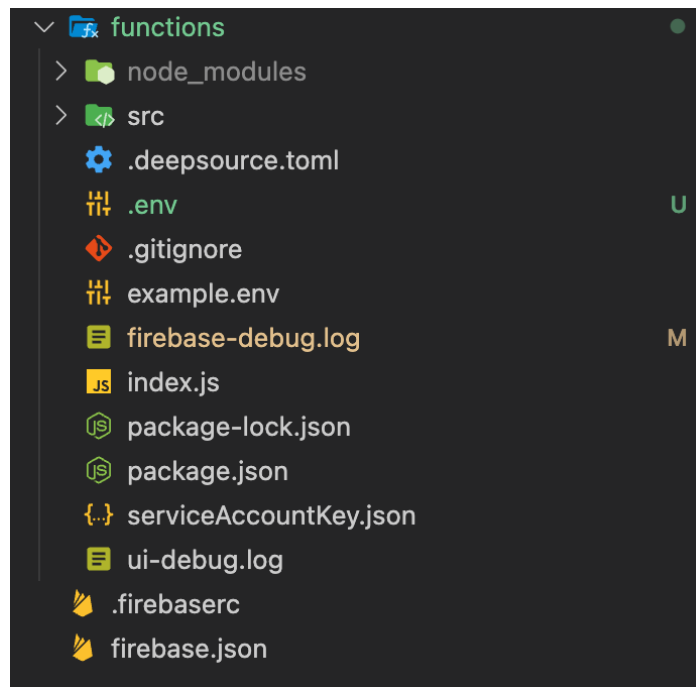
2. Backend

backend โดยระบบของ NFT-Marketplace ใช้ node Express และใช้ Firebase เป็น database โดยในโครงสร้างของระบบ backend ที่ใช้ express นั้นจะมีโครงสร้างดังนี้



รูปที่ 13 folder function

ซึ่งจะเห็นได้ว่ามี folder functions, ไฟล์ .firebaseconfig และไฟล์ firebase.json โดยระบบเราทำการ deploy ไว้ในระบบของ firebase functions ดังนั้นจึงต้องจัดการ structure ของระบบให้เป็นในรูปแบบนี้ เมื่อทำการเปิด folder functions ดังภาพนี้



รูปที่ 14 โครงสร้าง functions

ก็จะเห็นว่าใน folder ของ functions นั้นจะมี folder src ที่จะป็น code สำหรับการทํางานทั้งหมดของระบบ ไฟล์ .env ที่จะมีตัวแปร PRIVATE_KEY ที่ใช้สำหรับการอ่านค่าที่อยู่ในระบบของ blockchain และไฟล์ index.js

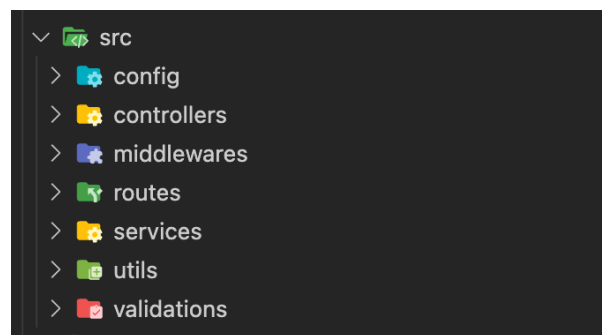
```

index.js
functions > index.js > ...
You, last month | 1 author (You)
1  const functions = require("firebase-functions");
2  const express = require("express");
3  const cors = require("cors");
4
5  const routes = require("./src/routes");
6
7  const app = express();
8  app.use(express.json());
9
10 app.use(
11   cors({
12     origin: "https://nft-marketplace-frontend.web.app",
13   })
14 );
15
16 app.get("/", (_, response) => {
17   response.status(200).send({ message: "NFT Marketplace!" });
18 });
19
20 app.use("/v1/", routes);
21
22 exports.app = functions.https.onRequest(app);
23

```

รูปที่ 15 index.js

ที่จะมีการกำหนด routes สำหรับการเรียกใช้งาน path แต่ละ path ที่ต้องการข้อมูลแตกต่างกันซึ่งจากภาพในบรรทัดที่ 20 มีการกำหนด path /v1/ และตามด้วย routes ซึ่งตัวของ routes เรียกใช้ routes จากไฟล์ path ./src/routes เมื่อเราเปิดเข้าไปใน folder src เพื่อจะไปที่ folder routes ที่อยู่ใน src จะเห็นโครงสร้างข้างใน src ดังนี้



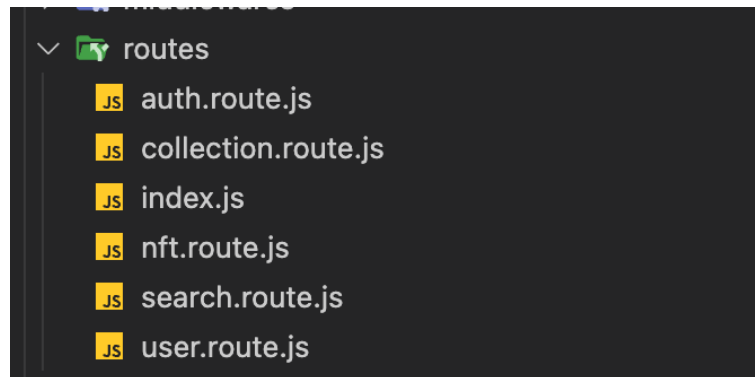
รูปที่ 16 โครงสร้าง src

ซึ่งใน folder src ก็จะมีการแบ่งโครงสร้างดังนี้

- 1) config
- 2) controllers
- 3) middlewares

- 4) routes
- 5) services
- 6) utils
- 7) validations

โดย folder ที่ถูกเรียกใช้เป็นอันดับแรกในไฟล์ index คือ routes เมื่อทำการเปิด folder routes จะเห็นไฟล์ทั้งหมดใน folder routes ดังนี้



รูปที่ 17 โครงสร้าง routes

จากภาพจะเห็นไฟล์ทั้งหมดใน routes ซึ่งการทำหลักจะอยู่ในไฟล์ index โดยหากเปิดไฟล์ index ขึ้นมาแล้วจะเห็น code ดังภาพ

```

functions > src > routes > js index.js > ...
You, 3 months ago | 2 authors (TNKKung and others)
1  const express = require("express");
2
3  const nftRoute = require("./nft.route");
4  const authRoute = require("./auth.route");
5  const collectionRoute = require("./collection.route");
6  const userRoute = require("./user.route");
7  const searchRoute = require("./search.route");
8
9  const router = express.Router();
10
11  const defaultRoutes = [
12    {
13      path: "/nft",
14      route: nftRoute,
15    },
16    {
17      path: "/auth",
18      route: authRoute,
19    },
20    {
21      path: "/collection",
22      route: collectionRoute,
23    },
24    {
25      path: "/user",
26      route: userRoute,
27    },
28    {
29      path: "/search",
30      route: searchRoute,
31    },
32  ];
33
34  defaultRoutes.forEach((route) => {
35    router.use(route.path, route.route);
36  });
37
38  module.exports = router;
39

```

รูปที่ 18 การเพิ่ม path

ตามภาพจะเห็นว่าการ เพิ่ม path สำหรับการ nft , auth, collection, user, search โดย path ที่สร้าง ขึ้นนี้ก็จะไปเรียกใช้ข้อมูลของ route ที่สร้างขึ้นมา nftRoute, authRoute, collectionRoute, userRoute, searchRoute ซึ่งหากผู้พัฒนาต้องการที่ติดต่อกับข้อมูลส่วนไหน ผู้พัฒนาก็ใส่ path ที่ต้องการใน API โดยแค่ route จะมีดังนี้

nftRoute แบ่งเป็น 4 ส่วน คือ Get, Post, Patch, Delete

1. API สำหรับการ Get ข้อมูล

```

router.get("/getAllTransaction", nftController.getAllTransaction);
router.get("/", nftController.getAllNFTsController);
router.get("/sale", nftController.getAllSaleNFTsController);
router.get("/random", nftController.getRandomNFTController);
router.get("/randomNFTSale", nftController.getRandomNFTSaleController);
router.get("/info", nftController.getInfoNFTsController);
router.get("/infoSale", nftController.getInfoSaleNFTsController);

router.get(
  "/getNFTByOwner",
  validate(nftValidation.getNFTByOwnerValidate),
  nftController.getNFTByOwnerController
);

router.get(
  "/getNFTCreatedByOwner",
  validate(nftValidation.getNFTCreatedByOwnerValidate),
  nftController.getNFTCreatedByOwnerController
);

router.get(
  "/getNFTByTokenId",
  validate(nftValidation.getNFTByTokenIdValidate),
  nftController.getNFTByTokenIdController
);

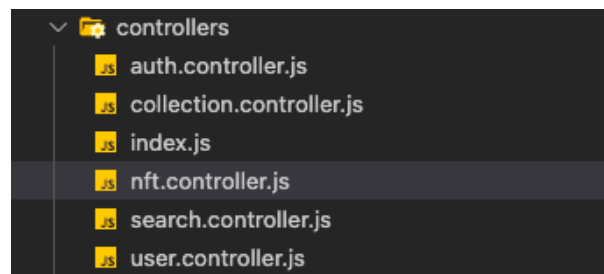
```

รูปที่ 19 โครงสร้าง router.get

โดยจะผู้พัฒนาจะเห็น router.get ซึ่งจะ get ข้อมูลตาม path ที่ทางผู้พัฒนาเลือกซึ่งแบ่งออกเป็น path ได้ดังนี้

- 1) Path /getAllTransaction โดย API นี้จะทำการเรียกใช้ module getAllTransaction ใน controller ของ NFT
- 2) Path / โดย API นี้จะทำการเรียกใช้ module getAllNFTsController ใน controller ของ NFT
- 3) Path /sale โดย API นี้จะทำการเรียกใช้ module getAllSaleNFTsController ใน controller ของ NFT
- 4) Path /random โดย API นี้จะทำการเรียกใช้ module getRandomNFTController ใน controller ของ NFT
- 5) Path /randomNFTSale โดย API จะทำการเรียกใช้ module getRandomNFTSaleController ใน controller ของ NFT
- 6) Path /info โดย API นี้จะทำการเรียกใช้ module getInfoNFTsController ใน controller ของ NFT

- 7) Path /infoSale โดย API นี้จะทำการเรียกใช้ module getInfoSaleNFTsController ใน controller ของ NFT
- 8) Path /getNFTByOwner โดย API นี้จะทำการเรียกใช้ module getNFTByOwnerController ใน controller ของ NFT และจะต้องมีการ validate ข้อมูล โดยการเรียกใช้ validate ของ NFT และใช้ module getNFTByOwnerValidate
- 9) Path /getNFTCreatedByOwner โดย API นี้จะทำการเรียกใช้ module getNFTCreatedByOwnerController ใน controller ของ NFT และจะต้องมีการ validate ข้อมูล โดยการเรียกใช้ validate ข้อมูลของ NFT และใช้ module getNFTCreatedByOwnerValidate
- 10) Path /getNFTByTokenId โดย API นี้ทำการเรียกใช้ module getNFTByTokenIdController ใน controller ของ NFT และจะต้องมีการ validate ข้อมูล โดยการเรียกใช้ validate ของ NFT และใช้ module getNFTByTokenIdValidate
- โดย controller ของ NFT ที่เรียกใช้นั้นจะอยู่ใน folder ของ controllers ดังภาพ



รูปที่ 20 controllers

และจะมี module สำหรับการ get ข้อมูลทั้งหมดดังภาพ

```

11 const getAllNFTsController = catchAsync(async (req, res) => {
12   const response = await nftService.getAllNFTService();
13   res.send({ response });
14 });
15
16 const getInfoNFTsController = catchAsync(async (req, res) => {
17   const response = await nftService.getInfoNFTService();
18   res.send({ response });
19 });
20
21 const getInfoSaleNFTsController = catchAsync(async (req, res) => {
22   const response = await nftService.getInfoSaleNFTService();
23   res.send({ response });
24 });
25
26 const getAllSaleNFTsController = catchAsync(async (req, res) => {
27   const response = await nftService.getAllSaleNFTService();
28   res.send({ response });
29 });
30
31 const getRandomNFTController = catchAsync(async (req, res) => {
32   const response = await nftService.getRandomNFTService();
33   res.send({ response });
34 });
35
36 const getRandomNFTSaleController = catchAsync(async (req, res) => {
37   const response = await nftService.getRandomNFTSaleService();
38   res.send({ response });
39 });
40
41 const getNFTByOwnerController = catchAsync(async (req, res) => {
42   const { address } = req.query;
43   const response = await nftService.getNFTByOwnerService(address);
44   res.send({ response });
45 });
46
47 const getNFTCreatedByOwnerController = catchAsync(async (req, res) => {
48   const { address } = req.query;
49   const response = await nftService.getNFTCreatedByOwnerService(address);
50   res.send({ response });
51 });
52
53 const getNFTByTokenIdController = catchAsync(async (req, res) => {
54   const { tokenId } = req.query;
55   const response = await nftService.getNFTByTokenIdService(tokenId);
56   res.send({ response });
57 });
58
59 const getAllTransaction = catchAsync(async (req, res) => {
60   const { id } = req.query;
61   const response = await nftService.getAllTransactionService(id);
62   res.send({ response });
63 });

```

You, 1 second ago • Uncommitted changes

รูปที่ 21 module สำหรับ get ข้อมูล

โดยแต่ละ module จะมีการทำงานดังนี้

- 1) getAllNFTsController โดย controller ตัวนี้จะมีการเรียกใช้งาน module getAllNFTService ที่เป็น service ของ NFT และทำการส่ง response กลับไป
- 2) getInfoNFTsController โดย controller ตัวนี้จะมีการเรียกใช้งาน module getInfoService ที่เป็น service ของ NFT และทำการส่ง response กลับไป

- 3) getInfoSaleNFTsController โดย controller ตัวนี้ จะมีการเรียกใช้งาน module getInfoSaleNFTService ที่เป็น service ของ NFT และทำการส่ง response กลับไป
- 4) getAllSaleNFTsController โดย controller ตัวนี้ จะมีการเรียกใช้งาน module getAllSaleNFTService ที่เป็น service ของ NFT และทำการส่ง response กลับไป
- 5) getRandomNFTController โดย controller ตัวนี้ จะมีการเรียกใช้งาน module getRandomNFTService ที่เป็น service ของ NFT และทำการส่ง response กลับไป
- 6) getRandomNFTSaleController โดย controller ตัวนี้ จะมีการเรียกใช้งาน module getRandomNFTSaleService ที่เป็น service ของ NFT และทำการส่ง response กลับไป
- 7) getNFTByOwnerController โดย controller ตัวนี้ จะมีการสร้างตัวแปร address สำหรับเก็บข้อมูลที่อยู่ใน query ของ request และนำไปใส่ใน module getNFTByOwnerService ที่เป็น service ของ NFT และทำการส่ง response กลับไป
- 8) getNFTCreatedByOwnerController โดย controller ตัวนี้ จะมีการสร้างตัวแปร address สำหรับเก็บข้อมูลที่อยู่ใน query ของ request และนำไปใส่ใน module getNFTCreatedByOwnerService ที่เป็น service ของ NFT และทำการส่ง response กลับไป
- 9) getNFTByTokenIdController โดย controller ตัวนี้ จะมีการสร้างตัวแปร tokenId สำหรับเก็บข้อมูลที่อยู่ใน query ของ request และนำไปใส่ใน module getNFTByTokenIdService ที่เป็น service ของ NFT และทำการส่ง response กลับไป
- 10) getAllTransaction โดย controller ตัวนี้ จะมีการสร้างตัวแปร id สำหรับเก็บข้อมูลที่อยู่ใน query ของ request และนำไปใส่ใน module getAllTransactionService ที่เป็น service ของ NFT และทำการส่ง response กลับไป

โดย controller เหล่านี้มีการเรียกใช้งาน service ที่มีการทำงานค้นหาข้อมูลใน database เพื่อที่จะทำการ return ข้อมูลกลับไปให้ controller ที่เรียกใช้ module ใน service เพื่อส่ง response กลับไปซึ่ง service ที่เป็น service สำหรับการ get ข้อมูลทั้งหมดดังนี้

```

91 > const getAllNFTService = async () => {--
97   };
98
99 > const getInfoNFTService = async () => {--
147   };
148
149 > const getAllSaleNFTService = async () => {--
157   };
158
159 > const getInfoSaleNFTService = async () => {--
208   };
209
210 > const getAllTransactionService = async (id) => { You, 2
255   };
256
257 > const getNFTByOwnerService = async (address) => {--
307   };
308
309 > const getNFTCreatedByOwnerService = async (address) => {--
359   };
360
361 > const getRandomNFTService = async (address) => {--
410   };
411
412 > const getRandomNFTSaleService = async () => {--
464   };
465
466 > const getNFTByTokenIdService = async (tokenId) => {--
507   };

```

รูปที่ 22 service สำหรับการ get

- 1) getAllNFTService เป็นการค้นหาข้อมูล NFT ทั้งหมดใน database และทำการ return ข้อมูลของ NFT ทั้งหมดไปให้ controller ที่เรียกใช้
- 2) getInfoNFTService เป็นการค้นหาข้อมูล NFT ทั้งหมดใน database แต่จะมีการคัดกรองข้อมูลที่สำคัญเช่น tokenId, nameNFT, tokenURI, collectionName, category, statusSale, price และทำการ return กลับไปให้ controller ที่เรียกใช้
- 3) getAllSaleNFTService เป็นการค้นหาข้อมูล NFT ทั้งหมดที่ทำการลงขายอยู่ในระบบ และทำการ return กลับไปให้ controller ที่เรียกใช้
- 4) getInfoSaleNFTService เป็นการค้นหาข้อมูลที่ NFT ทั้งที่ทำการลงขายอยู่ในระบบ พร้อมกับข้อมูลที่มีการคัดกรองข้อมูลที่สำคัญเช่น tokenId, nameNFT, tokenURI, collectionName, category, statusSale, price และทำการ return กลับไปให้ controller ที่เรียกใช้
- 5) getAllTransactionService เป็นการค้นหาข้อมูล transaction ของ NFT ที่ต้องการหาข้อมูลโดยจะมีการรับ id ของ token และจะทำการอ่านข้อมูล transactionHash และทำการอ่าน event และ return list ข้อมูล event, eventData, date, time กลับไปให้กลับ controller ที่เรียกใช้

- 6) `getNFTByOwnerService` เป็นการค้นหาข้อมูล NFT ทั้งหมดของ address ที่รับเข้ามา และทำการ return list ที่มีข้อมูล tokenId, nameNFT, tokenURI, collectionName, category, statusSale กลับไปให้ controller ที่เรียกใช้
 - 7) `getNFTCreatedByOwnerService` เป็นการค้นหาข้อมูล NFT ทั้งหมดที่สร้างโดย address ที่รับเข้ามาและทำการ return list ที่มีข้อมูล tokenId, nameNFT, tokenURI, collectionName, category, statusSale กลับไปให้ controller ที่เรียกใช้
 - 8) `getRandomNFTService` เป็นการค้นหาข้อมูล NFT แบบสุ่มและทำการ return NFT จำนวน 4 token ที่มีข้อมูลของ tokenId, nameNFT, tokenURI, collectionName, category, statusSale กลับไปให้ controller ที่เรียกใช้
 - 9) `getRandomNFTSaleService` เป็นการค้นหาข้อมูล NFT ที่ทำการลงขายแบบสุ่มและทำการ return NFT จำนวน 4 token ที่มีข้อมูลของ tokenId, nameNFT, tokenURI, collectionName, category, statusSale กลับไปให้ controller ที่เรียกใช้
 - 10) `getNFTByTokenIdService` เป็นการค้นหาข้อมูลของ NFT ที่ค้นหาจาก TokenId ที่รับเข้ามาและทำการ return ข้อมูลทั้งหมดจาก database กลับไปและมีการเพิ่ม id ,tokenURI และ price ของ token
2. API สำหรับการ POST

```
router.post(
  "/",
  jwtValidate,
  validate(nftValidation.createNFTValidate),
  nftController.createNFTController
);

router.post(
  "/addTransactionHash",
  jwtValidate,
  nftController.addTransactionHashController
);
```

รูปที่ 23 router.post

โดยจะผู้พัฒนาจะเห็น router.post ซึ่งจะ post ข้อมูลตาม path ที่ทางผู้พัฒนาเลือกซึ่งแบ่งออกเป็น path ได้ดังนี้

- 1) Path / โดย API นี้จะมีการเรียกใช้ module createNFTController ใน controller ของ NFT และ มีการเรียกใช้ middleware เพื่อทำการ validate jwt สำหรับการ post ข้อมูล และก็ เรียกใช้ module validate body โดยใช้ createNFTValidate
 - 2) Path /addTransactionHash โดย API นี้จะมีการเรียกใช้ module addTransactionHashController ใน controller ของ NFT และมีการเรียกใช้ middleware เพื่อทำการ validate jwt สำหรับการ post ข้อมูล
- โดยใน Route นี้จะมีการเรียกใช้งาน controller ของ NFT สำหรับการ post ดังนี้

```
const createNFTController = catchAsync(async (req, res) => {
  const response = await nftService.createNFTService(req.body);
  res.status(httpStatus.CREATED).send({ response });
});

const addTransactionHashController = catchAsync(async (req, res) => {
  const response = await nftService.addTransactionHashService(req.body);
  res.send({ response });
});
```

รูปที่ 24 createNFTController

- 1) createNFTController เป็นการเรียกใช้ module createNFTService ที่เป็น service ของ NFT และมี argument ใน module คือ req.body และจะได้ข้อมูลกลับมาเพื่อส่ง response กลับไป
- 2) addTransactionHashController เป็นการเรียกใช้ module addTransactionHashService ที่เป็น service ของ NFT และมี argument ใน module คือ req.body และจะได้ข้อมูลกลับมาเพื่อส่ง response กลับไป

โดย controller เหล่านี้มีการเรียกใช้งาน service ที่มีการเขียนข้อมูลลงไปใน database โดยจะมี module สำหรับ post ทั้งหมดดังนี้

```
21 > const createNFTService = async (body) => { ...
89   };
90
91 > const addTransactionHashService = async (body) => { ...
110  };
```

รูปที่ 25 module สำหรับ post

- 1) createNFTService เป็นการสร้าง NFT โดยที่จะรับ body เข้ามาและทำการสร้างตัวแปร tokenId,ownerAddress, nameNFT, description, category, collectionId, transactionHash และทำการเขียนข้อมูลลงไปใน database ดังนี้

```
const response = await storeNFTs.add({  
  ownerAddress: ownerAddress,  
  nameNFT: nameNFT,  
  description: description,  
  category: category,  
  collectionId: collectionId,  
  tokenId: tokenId,  
  transactionHash: [transactionHash],  
  createdCollaborator: result[0],  
  statusSale: false,  
  createdOwner: ownerAddress,  
});
```

รูปที่ 26 การเขียนข้อมูลลงใน database

- 2) addTransactionHashService เป็นการเพิ่มข้อมูลของ transactionHash เข้าไปใน list ของ transactionHash เข้าไปใน token ที่รับเข้ามา
3. API สำหรับ Patch

```
router.patch(  
  "/",  
  jwtValidate,  
  validate(nftValidation.updateCollectionOfNftValidate),  
  nftController.updateCollectionOfNFTController  
);  
  
router.patch(  
  "/updateOwner",  
  jwtValidate,  
  validate(nftValidation.updateOwnerNFTValidate),  
  nftController.updateOwnerNFTController  
);  
  
router.patch(  
  "/listingForSale",  
  jwtValidate,  
  validate(nftValidation.listingForSaleValidate),  
  nftController.listingForSaleController  
);  
  
router.patch(  
  "/unlistingForSale",  
  jwtValidate,  
  validate(nftValidation.unlistingForSaleValidate),  
  nftController.unlistingForSaleController  
);
```

รูปที่ 27 การ patch

โดยจะผู้พัฒนาจะเห็น router.patch ซึ่งจะ update ข้อมูลตาม path ที่ทางผู้พัฒนาเลือกซึ่งแบ่งออกเป็น path ได้ดังนี้

- 1) Path / จะมีการเรียกใช้ module updateCollectionOfNFTController ใน controller ของ NFT และมีการเรียกใช้ middleware เพื่อทำการ validate jwt สำหรับการ update ข้อมูล และมีการเรียกใช้ module validate ของ NFT ที่ validate body
- 2) Path /updateOwner จะมีการเรียกใช้ module updateOwnerNFTController ใน controller ของ NFT และมีการเรียกใช้ middleware เพื่อทำการ validate jwt สำหรับการ update ข้อมูล และมีการเรียกใช้ module validate ของ NFT ที่ validate body
- 3) Path /listingForSale จะมีการเรียกใช้ module listingForSaleController ใน controller ของ NFT และมีการเรียกใช้ middleware เพื่อทำการ validate jwt สำหรับการ update ข้อมูล และมีการเรียกใช้ module validate ของ NFT ที่ validate body และ validate query
- 4) Path /unlistingForSale จะมีการเรียกใช้ module unlistingForSaleController ใน controller ของ NFT และมีการเรียกใช้ middleware เพื่อทำการ validate jwt สำหรับการ update ข้อมูล และมีการเรียกใช้ module validate ของ NFT ที่ validate body และ validate query

โดย routes สำหรับการ update เหล่านี้เรียกใช้ controller ดังภาพ

```
const listingForSaleController = catchAsync(async (req, res) => {
  const { id } = req.body;
  const { ownerAddress } = req.query;
  const response = await nftService.listingForSaleService(id, ownerAddress);
  res.send({ response });
});

const unlistingForSaleController = catchAsync(async (req, res) => {
  const { id } = req.body;
  const { ownerAddress } = req.query;
  const response = await nftService.unlistingForSaleService(id, ownerAddress);
  res.send({ response });
});

const updateCollectionOfNFTController = catchAsync(async (req, res) => {
  const response = await nftService.updateCollectionOfNftService(req.body);
  res.send({ response });
});

const updateOwnerNFTController = catchAsync(async (req, res) => {
  const response = await nftService.updateOwnerNFTService(req.body);
  res.send({ response });
});
```

รูปที่ 28 routes สำหรับการ update

- 1) listingForSaleController รับ request body มาเป็น id และรับ request query มาเป็น ownerAddress และทำการเรียกใช้ module listingForSaleService ซึ่งเป็น service ของ NFT และมี argument ต้องใส่คือ id และ ownerAddress และทำการส่ง response กลับไป
- 2) unlistingForSaleController รับ request body มาเป็น id และรับ request query มาเป็น ownerAddress และทำการเรียกใช้ module unlistingForSaleService ซึ่งเป็น service ของ NFT และมี argument ต้องใส่คือ id และ ownerAddress และทำการส่ง response กลับไป
- 3) updateCollectionOfNFTController โดย controller นี้มีการเรียกใช้ module updateCollectionOfNftService และมี argument เป็น request body ที่ใส่เข้าไป และทำการส่ง response กลับไป
- 4) updateOwnerNFTController โดย controller นี้มีการเรียกใช้ module updateOwnerNFTService และมี argument เป็น request body ที่ใส่เข้าไป และทำการส่ง response กลับไป

โดย controller เหล่านี้มีการเรียกใช้ service ที่ใช้สำหรับการอัปเดตทั้งหมดดังนี้

```
> const listingForSaleService = async (id, ownerAddress) => {--
};

> const unlistingForSaleService = async (id, ownerAddress) => {--
};

> const updateCollectionOfNftService = async (body) => {--
};

> const updateOwnerNFTService = async (body) => {--
};
```

รูปที่ 29 routes สำหรับการ update

- 1) listingForSaleService การทำงานของ service นี้จะทำการรับ id ของ token และ ownerAddress เข้ามาเพื่อที่จะทำการเปลี่ยน status การลงขายของ NFT และ return string กลับไปยืนยันการ update
- 2) unlistingForSaleService การทำงานของ service นี้จะทำการรับ id ของ token และ ownerAddress เข้ามาเพื่อที่จะทำการยกเลิกการลงขายของ NFT และ return string กลับไปยืนยันการ update

- 3) updateCollectionOfNftService การทำงานของ service นี้จะทำการรับ id, collectionId, ownerAddress โดยจะมีการอัปเดต collection จาก collectionId ที่รับเข้ามา และ return string กลับไปขึ้นชั้นการ update
 - 4) updateOwnerNFTService การทำงานของ service นี้จะทำการรับ id จาก body โดยจะมีการอัปเดตเจ้าของ NFT ที่รับเข้ามา และ return string กลับไปขึ้นชั้นการ update
4. API สำหรับการ Delete

```
router.delete(
  "/",
  jwtValidate,
  validate(nftValidation.deleteNFTByTokenIdValidate),
  nftController.deleteNFTByTokenIdController
);
```

รูปที่ 30 routes สำหรับการ delete

โดยจะผู้พัฒนาจะเห็น router.delete ซึ่งจะ delete ข้อมูลตาม path ที่ทางผู้พัฒนาเลือกซึ่งแบ่งออกเป็น path ได้ดังนี้

- 1) Path / จะมีการเรียกใช้ module deleteNFTByTokenIdController ใน controller ของ NFT และมีการเรียกใช้ middleware เพื่อทำการ validate jwt สำหรับการ delete ข้อมูล และมีการเรียกใช้ module validate ของ NFT ที่ validate query โดย route นี้มีการเรียกใช้ controller สำหรับการ delete ข้อมูลดังนี้

```
const deleteNFTByTokenIdController = catchAsync(async (req, res) => {
  const { id } = req.query;
  const response = await nftService.deleteNFTByTokenIdService(id);
  res.send({ response });
});
```

รูปที่ 31 controller สำหรับการ delete NFTByTokenId

- 2) deleteNFTByTokenIdController โดยจะมีการรับ id ที่อยู่ใน request body และทำการเรียกใช้ module deleteNFTTokenService และใส่ argument เป็น id เข้าไปและจะได้ response ส่งกลับไปที่ controller นี้เรียกใช้ service สำหรับการ delete ข้อมูลที่อยู่ใน database ดังนี้

```

const deleteNFTByTokenIdService = async (tokenId) => {
  await storeNFTs.doc(tokenId).delete();
  return "delete NFT Success";
};

```

รูปที่ 32 controller สำหรับการ กำำ NFTByTokenId

- 3) deleteNFTByTokenIdService โดย Service นี้จะทำการลบข้อมูลใน database จาก tokenId ที่รับเข้ามาและทำการ return string เพื่อยืนยันการลบข้อมูล โดยจาก route ของ NFT มีการเรียกใช้งาน validation ที่เรียกใช้ nftValidation ในบรรทัดที่ 4 ดังภาพ

```

4 const { nftValidation } = require("../validations");

```

รูปที่ 33 nftValidation

จากบรรทัดที่ผู้พัฒนาจะเห็นได้ว่าการเรียกจาก folder validations ที่ไฟล์ nft.validation.js ดัง

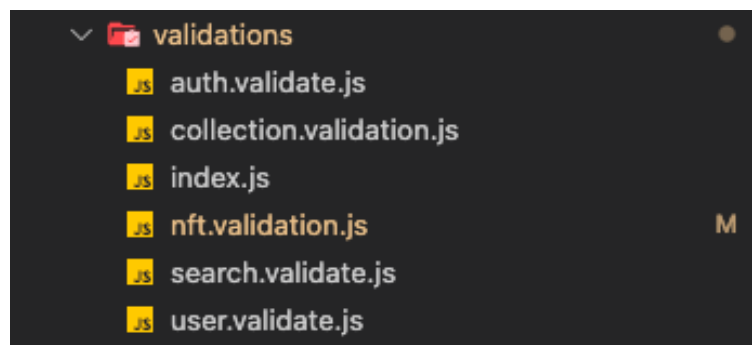
ภาพ

```

You, 2 months ago | 2 authors (TNKKung and others)
1 module.exports.nftValidation = require("../nft.validation");
2 module.exports.collectionValidation = require("../collection.validation");
3 module.exports.authValidation = require("../auth.validate");
4 module.exports.searchValidation = require("../search.validate");
5 module.exports.userValidation = require("../user.validate");
6

```

รูปที่ 34 folder validations



รูปที่ 35 โครงสร้าง validation

เมื่อผู้พัฒนาเข้าไปในไฟล์ nft.validation.js จะเห็นการสร้างตัว validate ข้อมูลดังนี้

```
3  const createNFTValidate = {
4    body: joi.object().keys({
5      ownerAddress: joi.string().required(),
6      nameNFT: joi.string().required(),
7      description: joi.string().required(),
8      category: joi.array().required(),
9      collectionId: joi.string().required(),
10     tokenId: joi.number().required(),
11     transactionHash: joi.string().required(),
12   }),
13 };
14
15 const getNFTByOwnerValidate = {
16   query: joi.object().keys({
17     address: joi.string().required(),
18   }),
19 };
20
21 const getNFTCreatedByOwnerValidate = {
22   query: joi.object().keys({
23     address: joi.string().required(),
24   }),
25 };
26
27 const getNFTByTokenIdValidate = {
28   query: joi.object().keys({
29     tokenId: joi.number().required(),
30   }),
31 };
```

รูปที่ 36 การสร้าง validate

```

33  const listingForSaleValidate = {
34    query: joi.object().keys({
35      ownerAddress: joi.string().required(),
36    }),
37    body: joi.object().keys({
38      id: joi.string().required(),
39    }),
40  };
41
42  const unlistingForSaleValidate = {
43    query: joi.object().keys({
44      ownerAddress: joi.string().required(),
45    }),
46    body: joi.object().keys({
47      id: joi.string().required(),
48    }),
49  };
50
51  const updateCollectionOfNftValidate = {
52    body: joi.object().keys({
53      id: joi.string().required(),
54      collectionId: joi.string().required(),
55    }),
56  };
57
58  const updateOwnerNFTValidate = {
59    body: joi.object().keys({
60      id: joi.string().required(),
61      contract: joi.string().required(),
62    }),
63  };
64
65  const deleteNFTByTokenIdValidate = {
66    query: joi.object().keys({
67      id: joi.string().required(),
68    }),
69  };

```

รูปที่ 37 listingForSaleValidate

โดยผู้พัฒนาจะสังเกตเห็นว่าตัวแปร Object ทั้งหมดโดยจะมีการแบ่งดังนี้

- 1) createNFTValidate ที่มีการสร้างว่า body จะต้องเป็น object ที่มี
 - 1) ownerAddress เป็น string
 - 2) nameNFT เป็น string
 - 3) description เป็น string
 - 4) category เป็น array
 - 5) collectionId เป็น string
 - 6) tokenId เป็น number
 - 7) transactionHash เป็น string
 - 2) getNFTByOwnerValidate ที่มีการสร้าง query ที่เป็น object ที่มี address เป็น string
 - 3) getNFTCreatedByOwnerValidate ที่มีการสร้าง query ที่เป็น object ที่มี address เป็น string
 - 4) getNFTByTokenIdValidate ที่มีการสร้าง query ที่เป็น object ที่มี tokenId เป็น number
 - 5) listingForSaleValidate ที่มีการสร้าง query ที่เป็น object ที่มี ownerAddress เป็น string และมีการสร้าง body ที่เป็น object ที่มี id เป็น string
 - 6) unlistingForSaleValidate ที่มีการสร้าง query ที่เป็น object ที่มี ownerAddress เป็น string และมีการสร้าง body ที่เป็น object ที่มี id เป็น string
 - 7) updateCollectionOfNftValidate ที่มีการสร้าง body ที่เป็น object ที่มี
 - 1) id เป็น string
 - 2) collectionId เป็น string
 - 8) updateOwnerNFTValidate ที่มีการสร้าง body ที่เป็น object ที่มี
 - 1) id เป็น string
 - 2) contract เป็น string
 - 9) deleteNFTByTokenIdValidate ที่มีการสร้าง query ที่เป็น object ที่มี id เป็น string
- authRoute จะมีเฉพาะ router get ดังภาพ

```

router.get(
  "/message",
  validate(authValidation.loginMessageValidate),
  authController.messageController
);

router.get(
  "/jwt",
  validate(authValidation.authJWTValidate),
  authController.authJWTController
);

router.get(
  "/requestAccessToken",
  jwtValidate,
  validate(authValidation.requestAccessTokenValidate),
  authController.requestAccessToken
);

```

รูปที่ 38 authRoute

API สำหรับการ Get มีดังนี้

- 1) Path /message จะมีการเรียกใช้ module messageController ใน controller ของ auth และมีการเรียกใช้ module loginMessageValidate ซึ่งเป็น validate ของ auth
- 2) Path /jwt จะมีการเรียกใช้ module authJWTController ใน controller ของ auth และมีการเรียกใช้ module authJWTValidate ซึ่งเป็น validate ของ auth
- 3) Path /requestAccessToken จะมีการเรียกใช้ module requestAccessToken ใน controller ของ auth และมีการเรียกใช้ module requestAccessTokenValidate ซึ่งเป็น validate ของ auth

โดย route นี้มีการเรียกใช้ controller สำหรับการ get ข้อมูลดังนี้

```
const messageController = catchAsync(async (req, res) => {
  const { address } = req.query;
  const response = await authService.loginMessage(address);
  res.send({ response });
});

const authJWTController = catchAsync(async (req, res) => {
  const { address, signature } = req.query;
  const response = await authService.authJWT(address, signature);
  res.send({ response });
});

const requestAccessToken = catchAsync(async (req, res) => {
  const { address } = req.query;
  const response = await authService.requestAccessToken(address);
  res.send({ response });
});
```

รูปที่ 39 messageController

- 1) messageController โดยจะมีการรับ address ที่อยู่ใน request query และทำการเรียกใช้ module loginMessage ซึ่งเป็น service ของ auth และใส่ argument เป็น address และส่ง response กลับไป
- 2) authJWTController โดยจะมีการรับ address , signature ที่อยู่ใน request query และทำการเรียกใช้ module authJWT ซึ่งเป็น service ของ auth และใส่ argument เป็น address , signature และส่ง response กลับไป
- 3) requestAccessToken โดยจะมีการรับ address ที่อยู่ใน request query และทำการเรียกใช้ module requestAccessToken ซึ่งเป็น service ของ auth และใส่ argument เป็น และส่ง response กลับไป

โดย controller เหล่านี้มีการเรียกใช้ service ที่ใช้สำหรับการ get ทั้งหมดดังนี้

```
> const loginMessage = async (address) => { ...
};

> const authJWT = async (address, signature) => { ...
};

> const requestAccessToken = async (address) => { ...
};
```

รูปที่ 40 service ที่ใช้ในการ get

- 1) loginMessage จะมีการทำงานคือการ getMessage ที่จะ sign กับ metamask หากมีข้อมูล user อยู่แล้วแต่หากไม่มีก็จะทำการเพิ่มข้อมูลลงไปใน database ของ user

```
admin.firestore().collection("Users").doc(address).set({
  address: address,
  name: address,
  bio: "",
  twitter: "",
  instagram: "",
  contact: "",
  profileImage: "",
  backgroundImage: "",
  messageToSign,
  friendList: [],
  favoriteNFT: [],
}, {
  merge: true,
});
```

รูปที่ 41 การเพิ่ม database ของ user

- 2) authJWT จะมีการรับ address เข้ามาและทำการ generate jwt โดย address และจะได้ access token และ refresh token

```
jwtGenerate(address),
jwtRefreshTokenGenerate(address),
```

รูปที่ 42 การ generate jwt

และทำการ return token ทั้งสองกลับไป

- 3) requestAccessToken จะมีการรับ address เข้ามาและทำการ generate access token อันใหม่และทำการ return ข้อมูลกลับไปโดย route ของ auth นั้นจะมีการเรียกใช้ authvalidation ดังรูป

```
const { authValidation } = require("../validations");
```

รูปที่ 43 authValidation

ซึ่งใน authValidation นั้นจะมีการ validate ดังนี้

```
const loginMessageValidate = {
  query: joi.object().keys({
    address: joi.string().required(),
  }),
};

const authJWTValidate = {
  query: joi.object().keys({
    address: joi.string().required(),
    signature: joi.string().required(),
  }),
};

const requestAccessTokenValidate = {
  query: joi.object().keys({
    address: joi.string().required(),
  }),
};
```

รูปที่ 44 validate ใน authValidation

- 1) loginMessageValidate ที่มีการสร้าง query จะต้องเป็น object ที่มี address เป็น string
- 2) authJWTValidate ที่มีการสร้าง query จะต้องเป็น object ที่มี
 - 1) address เป็น string
 - 2) Signature เป็น string
- 3) requestAccessTokenValidate ที่เป็นการสร้าง query จะต้องเป็น object ที่มี address เป็น string

collectionRoute จะแบ่งออกเป็น Get, Post , Delete, Patch ดังภาพ

```
router.get("/length", collectionController.getLengthCollectionController);

router.get(
  "/",
  validate(collectionValidation.getAllCollectionValidate),
  collectionController.getAllCollectionController
);

router.get("/random", collectionController.getCollectionExploreController);

router.get(
  "/getCollectionById",
  validate(collectionValidation.getCollectionByIdValidate),
  collectionController.getCollectionByIdController
);

router.get(
  "/getCollectionByOwner",
  validate(collectionValidation.getCollectionByOwnerValidate),
  collectionController.getCollectionByOwnerController
);

router.post(
  "/",
  jwtValidate,
  validate(collectionValidation.createCollectionValidate),
  collectionController.createCollectionController
);

router.delete(
  "/",
  jwtValidate,
  validate(collectionValidation.deleteCollectionByIdValidate),
  collectionController.deleteCollectionByIdController
);

router.patch(
  "/",
  jwtValidate,
  validate(collectionValidation.updateCollectionValidate),
  collectionController.updateCollectionController
);
```

รูปที่ 45 collectionRoute

API สำหรับการ Get มีดังนี้

- 1) Path /length จะมีการเรียกใช้ module getLengthCollectionController ใน controller ของ collection
- 2) Path / จะมีการเรียกใช้ module getAllCollectionController ใน controller ของ collection และมีการเรียกใช้ module getAllCollectionValidate ซึ่งเป็น validate ของ collection
- 3) Path /random จะมีการเรียกใช้ module getCollectionExploreController ใน controller ของ collection

- 4) Path /getCollectionById จะมีการเรียกใช้ module getCollectionByIdController ใน controller ของ collection และมีการเรียกใช้ module getCollectionByIdValidate ซึ่งเป็น validate ของ collection
- 5) Path /getCollectionByOwner จะมีการเรียกใช้ module getCollectionByOwnerController ใน controller ของ collection และมีการเรียกใช้ module getCollectionByOwnerValidate ซึ่งเป็น validate ของ collection
- โดยใน route ของ collection นั้นจะมีการเรียกใช้ controller สำหรับการ get ข้อมูลดังนี้

```
✓ const getAllCollectionController = catchAsync(async (req, res) => {  
  const response = await collectionService.getAllCollectionService();  
  res.status(200).send({ response });  
});  
  
✓ const getLengthCollectionController = catchAsync(async (req, res) => {  
  const response = await collectionService.getLengthCollection();  
  res.status(200).send({ response });  
});  
  
✓ const getCollectionExploreController = catchAsync(async (req, res) => {  
  const response = await collectionService.getAllExploreCollectionService();  
  res.send({ response });  
});  
  
✓ const getCollectionByIdController = catchAsync(async (req, res) => {  
  ✓ const response = await collectionService.getCollectionByIdService(  
    req.query.id  
  );  
  res.status(200).send({ response });  
});  
  
✓ const getCollectionByOwnerController = catchAsync(async (req, res) => {  
  ✓ const response = await collectionService.getCollectionByOwnerService(  
    req.query.owner  
  );  
  res.status(200).send({ response });  
});
```

รูปที่ 46 controller สำหรับการ get

- 1) getAllCollectionController โดยจะมีการเรียกใช้ module getAllCollectionService ซึ่งเป็น service ของ collection และทำการส่ง response กลับไป
- 2) getLengthCollectionController โดยจะมีการเรียกใช้ module getLengthCollection ซึ่งเป็น service ของ collection และทำการส่ง response กลับไป
- 3) getCollectionExploreController โดยจะมีการเรียกใช้ module getAllExploreCollectionService ซึ่งเป็น service ของ collection และทำการส่ง response กลับไป

- 4) `getCollectionByIdController` โดยจะมีการรับตัวแปร `id` จาก request query และมีการเรียกใช้ module `getCollectionByIdService` และทำการใส่ใน argument คือ `id` โดย module นี้เป็น service ของ collection และทำการส่ง response กลับไป
- 5) `getCollectionByOwnerController` โดยจะมีการรับตัวแปร `owner` จาก request query และมีการเรียกใช้ module `getCollectionByOwnerService` และทำการใส่ใน argument คือ `owner` โดย module นี้เป็น service ของ collection และทำการส่ง response กลับไปซึ่ง controller ก็ได้มีการเรียกใช้ข้อมูลใน service ของ collection ดังนี้

```
> const getLengthCollection = async () => {--  
};  
  
> const getAllCollectionService = async () => {--  
};  
  
> const getAllExploreCollectionService = async () => {--  
};  
  
> const getCollectionByIdService = async (id) => {--  
};  
  
> const getCollectionByOwnerService = async (owner) => {--  
};
```

รูปที่ 47 service ของ collection

- 1) `getLengthCollection` โดย Service นี้จะมีการ get ข้อมูลของ collection โดยจะส่งกลับไปเป็น length ความยาวของ collection
- 2) `getAllCollectionService` โดย service นี้จะมีการ get ข้อมูลทั้งหมดของ collection ทุก collection และมีเพิ่ม nft image เข้าไปด้วย
- 3) `getAllExploreCollectionService` โดย service นี้จะมีการ get ข้อมูลแบบสุ่มเพื่อส่งข้อมูลของ collection จำนวน 4 collection พร้อมกับ nft image กลับไปด้วย
- 4) `getCollectionByIdService` จะมีการรับ `id` เข้ามาเพื่อใช้ในการค้นหา collection ที่ตรงกับ `id` และทำการ return ข้อมูลของ collection นั้นกลับไป
- 5) `getCollectionByOwnerService` จะมีการรับ address ของ owner เข้ามาและหา collection ทั้งหมดของ owner เจ้าของ address และทำการ return list ของ collection กลับไป

API สำหรับการ Post มีดังนี้

- 1) Path / จะมีการเรียกใช้ module createCollectionController ใน controller ของ collection และมีการเรียกใช้ module createCollectionValidate ซึ่งเป็น validate ของ collection และมีการ validate jwt สำหรับการเขียนข้อมูลลง database โดย path นี้มีการเรียกใช้งาน controller สำหรับการ post ข้อมูลดังนี้

```
const createCollectionController = catchAsync(async (req, res) => {
  const response = await collectionService.createCollectionService(req.body);
  res.status(HttpStatus.CREATED).send({ response });
});
```

รูปที่ 48 createCollectionController

- 2) createCollectionController โดยจะมีการรับตัวแปรจาก request body และมีการเรียกใช้ module createCollectionService และทำการใส่ใน argument คือ request body โดย module นี้เป็น service ของ collection และทำการส่ง response กลับไปซึ่ง controller ก็ได้มีการเรียกใช้ข้อมูลใน service ของ collection ดังนี้

```
const createCollectionService = async (body) => {
  const response = await storeCollection.add({
    owner: body.owner,
    collectionName: body.collectionName,
    description: body.description,
  });
  const data = await storeCollection.doc(response.id).get();
  if (!data.exists) {
    console.log("No such document!");
  } else {
    const info = {
      collectionId: response.id,
      owner: data.data().owner,
      collectionName: data.data().collectionName,
      description: data.data().description,
    };
    await store.collection("Collections").doc(response.id).set(info);
    return info;
  }
};
```

รูปที่ 49 createCollectionService

- 3) createCollectionService โดยจะมีการรับ body ที่มี owner, collectionName, description เข้ามาและทำการสร้าง collection และตามด้วยการเพิ่มข้อมูลของ collectionId เข้าไป และ return ข้อมูลที่สร้างขึ้นมากลับไป

API สำหรับการ Delete มีดังนี้

- 1) Path / จะมีการเรียกใช้ module deleteCollectionByIdController ใน controller ของ collection และมีการเรียกใช้ module deleteCollectionByIdValidate ซึ่งเป็น validate ของ collection และมีการ validate jwt สำหรับการลบข้อมูลออกจาก database โดย path นี้มีการเรียกใช้งาน controller สำหรับการ Delete ข้อมูลดังนี้

```
const deleteCollectionByIdController = catchAsync(async (req, res) => {  
  const { id } = req.query;  
  const response = await collectionService.deleteCollectionByIdService(id);  
  res.status(200).send({ response });  
});
```

รูปที่ 50 deleteCollectionByIdController

- 2) deleteCollectionByIdController โดยจะมีการรับตัวแปร id จาก request query และมีการเรียกใช้ module deleteCollectionByIdService และทำการใส่ใน argument คือ id โดย module นี้เป็น service ของ collection และทำการส่ง response กลับไปซึ่ง controller ก็ได้มีการเรียกใช้ข้อมูลใน service ของ collection ดังนี้

```
const deleteCollectionByIdService = async (id) => {  
  const NFTs = await storeNFT.get();  
  NFTs.docs.map(async (doc) => {  
    if (doc.data().collectionId === id) {  
      const info = {  
        numberAddress: doc.data().numberAddress,  
        (property) description: any  
        description: doc.data().description,  
        tokenId: doc.data().tokenId,  
        category: doc.data().category,  
        collectionId: "-",  
      };  
      await storeNFT.doc(doc.id).set(info);  
    }  
  });  
  await storeCollection.doc(id).delete();  
  return "Delete collection success";  
};
```

รูปที่ 51 deleteCollectionByIdService

- 3) deleteCollectionByIdService โดยจะมีการรับ id เข้ามาเพื่อทำการค้นหา collection ที่ต้องการจะลบและทำการลบข้อมูลจาก database และทำการ return string กลับไปบอกผลการลบข้อมูล,

API สำหรับการ Patch มีดังนี้

- 1) Path / จะมีการเรียกใช้ module updateCollectionController ใน controller ของ collection และมีการเรียกใช้ module updateCollectionValidate ซึ่งเป็น validate ของ collection และมีการ validate jwt สำหรับการอัปเดตข้อมูลใน database โดย path นี้มีการเรียกใช้งาน controller สำหรับการ Patch ข้อมูลดังนี้

```
const updateCollectionController = catchAsync(async (req, res) => {  
  const response = await collectionService.updateCollectionService(req.body);  
  res.status(200).send({ response });  
});
```

รูปที่ 52 updateCollectionController

- 2) updateCollectionController โดยจะมีการรับตัวแปรจาก request body และมีการเรียกใช้ module updateCollectionService และทำการใส่ใน argument คือ request body โดย module นี้เป็น service ของ collection และทำการส่ง response กลับไปซึ่ง controller ก็ได้มีการเรียกใช้ข้อมูลใน service ของ collection ดังนี้

```
const updateCollectionService = async (body) => {  
  const data = await storeCollection.doc(body.id).get();  
  if (!data.exists) {  
    console.log("No such document!");  
  } else {  
    await storeCollection.doc(body.id).set({  
      collectionId: data.data().collectionId,  
      owner: data.data().owner,  
      collectionName: body.collectionName,  
      description: body.description,  
    });  
  }  
  return "update collection success";  
};
```

รูปที่ 53 updateCollectionService

- 3) updateCollectionService โดยจะมีการรับ id จาก body เพื่อใช้ค้นหา collection ที่ต้องการ จะ update ข้อมูล หากเจอจะทำการนำ collectionName, description จาก body เพื่อใช้อัปเดตข้อมูลใน collectionValidation นั้นจะมีการ validate ดังนี้


```

const createCollectionValidate = {
  body: joi.object().keys({
    owner: joi.string().required(),
    collectionName: joi.string().required(),
    description: joi.string().required(),
  }),
};

const getAllCollectionValidate = {
  body: joi.object().keys({}),
};

const getCollectionByIdValidate = {
  query: joi.object().keys({
    id: joi.string().required(),
  }),
};

const getCollectionByOwnerValidate = {
  query: joi.object().keys({
    owner: joi.string().required(),
  }),
};

const deleteCollectionByIdValidate = {
  query: joi.object().keys({
    id: joi.string().required(),
  }),
};

const updateCollectionValidate = {
  body: joi.object().keys({
    id: joi.string().required(),
    collectionName: joi.string().required(),
    description: joi.string(),
  }),
};

```

รูปที่ 54 createCollectionValidate

- 4) createCollectionValidate ที่มีการสร้าง body จะต้องเป็น object ที่มี
 - 1) owner เป็น string
 - 2) collectionName เป็น string
 - 3) description เป็น string
- 5) getAllCollectionValidate ที่มีการสร้าง query จะต้องเป็น object
- 6) getCollectionByIdValidate ที่เป็นการสร้าง query จะต้องเป็น object ที่มี owner เป็น string
- 7) deleteCollectionByIdValidate ที่เป็นการสร้าง body จะต้องเป็น object ที่มี
 - 1) id เป็น string
 - 2) collectionName เป็น string
 - 3) description เป็น string

userRoute จะแบ่งออกเป็น get, post ดังนี้

```
router.get(
  "/getUserByAddress",
  validate(userValidation.getUserByAddress),
  userController.getUserByAddressController
);

router.post(
  "/addFriendList",
  jwtValidate,
  validate(userValidation.addFriendList),
  userController.addFriendListController
);

router.post(
  "/unfriendList",
  jwtValidate,
  validate(userValidation.unfriendList),
  userController.unfriendListController
);

// You, 4 months ago * feat: add unfriend for user
router.post(
  "/addFavoriteNFT",
  jwtValidate,
  validate(userValidation.addFavoriteNFT),
  userController.addFavoriteNFTController
);

router.post(
  "/removeFavoriteNFT",
  jwtValidate,
  validate(userValidation.removeFavoriteNFT),
  userController.removeFavoriteNFTController
);

router.post(
  "/editInfoUser",
  jwtValidate,
  userController.editInfoUserController
);

router.post(
  "/editImageProfile",
  jwtValidate,
  validate(userValidation.editImageProfile),
  userController.editImageProfileController
);

router.post(
  "/editImageBackground",
  jwtValidate,
  validate(userValidation.editImageBackground),
  userController.editImageBackgroundController
);
```

รูปที่ 55 userRoute

API สำหรับการ get ข้อมูลดังภาพ

```
router.get(
  "/",
  validate(userValidation.getAllUsers),
  userController.getAllUserController
);

router.get(
  "/getUserByAddress",
  validate(userValidation.getUserByAddress),
  userController.getUserByAddressController
);
```

รูปที่ 56 api สำหรับการ get ข้อมูล

- 1) Path / โดย Service นี้จะมีการ get ข้อมูลของ user โดยจะมีการเรียกใช้ module getAllUserController ของ user controller และมีการเรียกใช้ validate สำหรับการ validate getAllUsers
 - 2) Path /getUserByAddress โดย Service นี้จะมีการ get ข้อมูลของ user โดยจะมีการเรียกใช้ module getUserByAddressController ของ user controller และมีการเรียกใช้ validate สำหรับการ validate getUserByAddress
- โดย path นี้มีการเรียกใช้งาน controller สำหรับการ get ข้อมูลดังนี้

```
const getAllUserController = catchAsync(async (req, res) => {
  const response = await userService.getAllUsersService();
  res.status(HttpStatus.CREATED).send({ response });
});

const getUserByAddressController = catchAsync(async (req, res) => {
  const { address } = req.query;
  const response = await userService.getUserByAddressService(address);
  res.send({ response });
});
```

รูปที่ 57 getAllUserController

- 1) getAllUserController จะมีการเรียกใช้ module getAllUsersService ซึ่งเป็น service ของ user และจะทำการส่ง response กลับไป
- 2) getUserByAddressController โดยจะมีการรับ address จาก request query และมีการเรียกใช้ module getUserByAddressService และทำการนำ address ใส่ใน argument โดย

module นี้เป็น service ของ user และทำการส่ง response กลับไป ซึ่ง controller ก็ได้มีการเรียกใช้ข้อมูลใน service ของ user ดังนี้

```
const getAllUsersService = async () => {  
  const users = await storeUsers.get();  
  
  return users.docs.map((doc) => {  
    return { ...doc.data() };  
  });  
};  
  
const getUserByAddressService = async (address) => {  
  const storeUser = await storeUsers.get();  
  
  const filterData = storeUser.docs.filter(  
    (doc) => doc.data().address === address  
  );  
  
  const data = filterData.map((data) => data.data());  
  
  return data[0];  
};
```

รูปที่ 58 getAllUserService

- 1) getAllUsersService จะมีการทำงานคือการ get ข้อมูลของ user ทั้งหมด และทำการ return ข้อมูลทั้งหมดกลับไป
- 2) getUserByAddressService จะมีการทำงานคือทำการรับ address เข้ามาเพื่อทำการ filter user ที่ตรงกับ address และทำการ return ข้อมูลกลับไป

API สำหรับการ post ข้อมูลคังภาพ

```
router.post(
  "/addFriendList",
  jwtValidate,
  validate(userValidation.addFriendList),
  userController.addFriendListController
);

router.post(
  "/unfriendList",
  jwtValidate,
  validate(userValidation.unfriendList),
  userController.unfriendListController
);

// You, 4 months ago * feat: add unfriend for

router.post(
  "/addFavoriteNFT",
  jwtValidate,
  validate(userValidation.addFavoriteNFT),
  userController.addFavoriteNFTController
);

router.post(
  "/removeFavoriteNFT",
  jwtValidate,
  validate(userValidation.removeFavoriteNFT),
  userController.removeFavoriteNFTController
);

router.post(
  "/editInfoUser",
  jwtValidate,
  userController.editInfoUserController
);

router.post(
  "/editImageProfile",
  jwtValidate,
  validate(userValidation.editImageProfile),
  userController.editImageProfileController
);

router.post(
  "/editImageBackground",
  jwtValidate,
  validate(userValidation.editImageBackground),
  userController.editImageBackgroundController
);
```

รูปที่ 59 path/unfriendList

- 1) Path /unfriendList จะมีการเรียกใช้ module unfriendListController ใน controller ของ user และมีการเรียกใช้ module unfriendList ซึ่งเป็น validate ของ user และมีการ validate jwt สำหรับการเขียนข้อมูลลง database

- 2) Path /addFavoriteNFT จะมีการเรียกใช้ module addFavoriteNFTController ใน controller ของ user และมีการเรียกใช้ module addFavoriteNFT ซึ่งเป็น validate ของ user และมีการ validate jwt สำหรับการเขียนข้อมูลลง database
- 3) Path /removeFavoriteNFT จะมีการเรียกใช้ module removeFavoriteNFTController ใน controller ของ user และมีการเรียกใช้ module removeFavoriteNFT ซึ่งเป็น validate ของ user และมีการ validate jwt สำหรับการเขียนข้อมูลลง database
- 4) Path /editInfoUser จะมีการเรียกใช้ module editInfoUserController ใน controller ของ user และมีการ validate jwt สำหรับการเขียนข้อมูลลง database
- 5) Path /editImageProfile จะมีการเรียกใช้ module editImageProfileController ใน controller ของ user และมีการเรียกใช้ module editImageProfile ซึ่งเป็น validate ของ user และมีการ validate jwt สำหรับการเขียนข้อมูลลง database
- 6) Path /editImageBackground จะมีการเรียกใช้ module editImageBackgroundController ใน controller ของ user และมีการเรียกใช้ module editImageBackground ซึ่งเป็น validate ของ user และมีการ validate jwt สำหรับการเขียนข้อมูลลง database โดย path นี้มีการเรียกใช้งาน controller สำหรับการ post ข้อมูลดังนี้

```

> const addFriendListController = catchAsync(async (req, res) => {
  });
> const unfriendListController = catchAsync(async (req, res) => {
  });
> const addFavoriteNFTController = catchAsync(async (req, res) => {
  });
> const removeFavoriteNFTController = catchAsync(async (req, res) => {
  });
> const editInfoUserController = catchAsync(async (req, res) => {
  });
> const editImageProfileController = catchAsync(async (req, res) => {
  });
> const editImageBackgroundController = catchAsync(async (req, res) => {
  });

```

รูปที่ 60 addFreindListController

- 1) addFriendListController โดยจะมีการรับตัวแปร address จาก request query และ frinedAddress จาก request body และมีการเรียกใช้ module addFrinendListService

และทำการใส่ใน argument คือ address , friendAddress โดย module นี้เป็น service ของ user และทำการส่ง response กลับไป

- 2) unfriendListController โดยจะมีการรับตัวแปร address จาก request query และ friendAddress จาก request body และมีการเรียกใช้ module unfriendListService และทำการใส่ใน argument คือ address , friendAddress โดย module นี้เป็น service ของ user และทำการส่ง response กลับไป
- 3) addFavoriteNFTController โดยจะมีการรับตัวแปร address จาก request query และมีการเรียกใช้ module addFavoriteNFTService และทำการใส่ใน argument คือ address , request body โดย module นี้เป็น service ของ user และทำการส่ง response กลับไป
- 4) removeFavoriteNFTController โดยจะมีการรับตัวแปร address จาก request query และ tokenId จาก request body และมีการเรียกใช้ module removeFavoriteNFTService และทำการใส่ใน argument คือ address , tokenId โดย module นี้เป็น service ของ user และทำการส่ง response กลับไป
- 5) editInfoUserController โดยจะมีการรับตัวแปร address จาก request query และมีการเรียกใช้ module editInfoUserService และทำการใส่ใน argument คือ address , request body โดย module นี้เป็น service ของ user และทำการส่ง response กลับไป
- 6) editImageProfileController โดยจะมีการรับตัวแปร address จาก request query และ profileImage จาก request body และมีการเรียกใช้ module editImageProfileService และทำการใส่ใน argument คือ address , profileImage โดย module นี้เป็น service ของ user และทำการส่ง response กลับไป
- 7) editImageBackgroundController โดยจะมีการรับตัวแปร address จาก request query และ backgroundImage จาก request body และมีการเรียกใช้ module editImageBackgroundService และทำการใส่ใน argument คือ address , backgroundImage โดย module นี้เป็น service ของ user และทำการส่ง response กลับไป

ใน userValidation นั้นจะมีการ validate ดังนี้

```

const getUserByAddressValidate = {
  query: joi.object().keys({
    address: joi.string().required(),
  }),
};

const addFriendListValidate = {
  query: joi.object().keys({
    address: joi.string().required(),
  }),
  body: joi.object().keys({
    friendAddress: joi.string().required(),
  }),
};

const unfriendListValidate = {
  query: joi.object().keys({
    address: joi.string().required(),
  }),
  body: joi.object().keys({
    friendAddress: joi.string().required(),
  }),
};

const addFavoriteNFTValidate = {
  query: joi.object().keys({
    address: joi.string().required(),
  }),
  body: joi.object().keys({
    tokenId: joi.number().required(),
  }),
};

const removeFavoriteNFTValidate = {
  query: joi.object().keys({
    address: joi.string().required(),
  }),
  body: joi.object().keys({
    tokenId: joi.number().required(),
  }),
};

const editInfoUserValidate = {
  query: joi.object().keys({
    address: joi.string().required(),
  }),
  body: joi.object().keys({
    name: joi.string().required(),
    bio: joi.string().required(),
    twitter: joi.string(),
    instagram: joi.string(),
    contact: joi.string(),
  }),
};

```

รูปที่ 61 getUserByAddressValidate

- 1) getUserByAddressValidate ที่มีการสร้าง query จะต้องเป็น object ที่มี address เป็น string
- 2) addFriendListValidate ที่มีการสร้าง query จะต้องเป็น object ที่มี
- 3) address เป็น string และสร้าง body ที่จะต้องเป็น object ที่มี
- 4) friendAddress เป็น string

- 5) unfriendListValidate ที่มีการสร้าง query จะต้องเป็น object ที่มี
- 6) address เป็น string และสร้าง body ที่จะต้องเป็น object ที่มี
- 7) friendAddress เป็น string
- 8) addFavoriteNFTValidate ที่มีการสร้าง query จะต้องเป็น object ที่มี
- 9) address เป็น string และสร้าง body ที่จะต้องเป็น object ที่มี
- 10) tokenId เป็น number
- 11) removeFavoriteNFTValidate ที่มีการสร้าง query จะต้องเป็น object ที่มี
- 12) address เป็น string และสร้าง body ที่จะต้องเป็น object ที่มี
- 13) tokenId เป็น number
- 14) editInfoUserValidate ที่มีการสร้าง query จะต้องเป็น object ที่มี
- 15) address เป็น string และสร้าง body ที่จะต้องเป็น object ที่มี
- 16) name เป็น string
- 17) bio เป็น string
- 18) twitter เป็น string
- 19) instagram เป็น string
- 20) contact เป็น string
- 21) editImageProfileValidate ที่มีการสร้าง query จะต้องเป็น object ที่มี
- 22) address เป็น string และสร้าง body ที่จะต้องเป็น object ที่มี
- 23) profileImage เป็น string
- 24) editImageBackgroundValidate ที่มีการสร้าง query จะต้องเป็น object ที่มี
- 25) address เป็น string และสร้าง body ที่จะต้องเป็น object ที่มี
- 26) backgroundImage เป็น string

searchRoute จะมีเฉพาะ router get ดังนี้

```
router.get(
  "/",
  validate(searchValidation.getAllSearchValidate),
  searchController.allSearchController
);

router.get(
  "/getNFTSearch",
  validate(searchValidation.getNFTsSearchValidate),
  searchController.nftsSearchController
);

router.get(
  "/getUserSearch",
  validate(searchValidation.getUsersSearchValidate),
  searchController.usersSearchController
);

router.get(
  "/getCollectionSearch",
  validate(searchValidation.getCollectionsSearchValidate),
  searchController.collectionsSearchController
);
```

รูปที่ 62 path โดย Service

- 1) Path / โดย Service นี้จะมีการ get ข้อมูลทั้งหมดของการ search ทั้ง collection, user, NFT โดยจะมีการเรียกใช้ module allSearchController ของ search controller และมีการเรียกใช้ validate สำหรับการ validate getAllSearchValidate
 - 2) Path /getNFTSearch โดย Service นี้จะมีการ get ข้อมูลของการ search โดยจะมีการเรียกใช้ module nftsSearchController ของ search controller และมีการเรียกใช้ validate สำหรับการ validate getNFTsSearchValidate
 - 3) Path /getUserSearch โดย Service นี้จะมีการ get ข้อมูลของการ search โดยจะมีการเรียกใช้ module usersSearchController ของ search controller และมีการเรียกใช้ validate สำหรับการ validate getUsersSearchValidate
 - 4) Path /getCollectionSearch โดย Service นี้จะมีการ get ข้อมูลของการ search โดยจะมีการเรียกใช้ module collectionSearchController ของ search controller และมีการเรียกใช้ validate สำหรับการ validate getCollectionSearchValidate
- โดย path นี้มีการเรียกใช้งาน controller สำหรับการ get ข้อมูลดังนี้

```

const allSearchController = catchAsync(async (req, res) => {
  const { keyword } = req.query;
  const response = await searchService.getAllSearchListService(keyword);

  res.send({ response });
});

const nftsSearchController = catchAsync(async (req, res) => {
  const { keyword } = req.query;
  const response = await searchService.getNFTsSearchService(keyword);

  res.send({ response });
});

const usersSearchController = catchAsync(async (req, res) => {
  const { keyword } = req.query;
  const response = await searchService.getUsersSearchService(keyword);

  res.send({ response });
});

const collectionsSearchController = catchAsync(async (req, res) => {
  const { keyword } = req.query;
  const response = await searchService.getCollectionSearchService(keyword);

  res.send({ response });
});

```

รูปที่ 63 allSearchController

- 1) allSearchController โดยจะมีการรับ keyword จาก request query และจะมีการเรียกใช้ module getAllSearchListService และทำการนำ keyword ใส่นำใน argument โดย module นี้เป็น service ของ search และทำการส่ง response กลับไป
- 2) nftsSearchController โดยจะมีการรับ keyword จาก request query และจะมีการเรียกใช้ module getNFTsSearchService และทำการนำ keyword ใส่นำใน argument โดย module นี้เป็น service ของ search และทำการส่ง response กลับไป
- 3) usersSearchController โดยจะมีการรับ keyword จาก request query และจะมีการเรียกใช้ module getUsersSearchService และทำการนำ keyword ใส่นำใน argument โดย module นี้เป็น service ของ search และทำการส่ง response กลับไป
- 4) collectionsSearchController โดยจะมีการรับ keyword จาก request query และจะมีการเรียกใช้ module getCollectionsSearchService และทำการนำ keyword ใส่นำใน argument โดย module นี้เป็น service ของ search และทำการส่ง response กลับไป

ซึ่ง controller ก็ได้มีการเรียกใช้ข้อมูลใน service ของ user ดังนี้

```
> const getAllSearchListService = async (keyword) => {--  
};  
  
> const getNFTsSearchService = async (keyword) => {--  
};  
  
> const getUsersSearchService = async (keyword) => {--  
};  
  
> const getCollectionSearchService = async (keyword) => {--  
};
```

รูปที่ 64 controller service

- 1) getAllSearchListService จะมีการทำงานคือการ get ข้อมูลของการ search ข้อมูลทั้ง NFT, user, collection และทำการ return ข้อมูลทั้งหมดกลับไป
- 2) getNFTsSearchService จะมีการทำงานคือการ get ข้อมูลของการ search ข้อมูลทั้ง NFT และทำการ return ข้อมูลทั้งหมดกลับไป
- 3) getUsersSearchService จะมีการทำงานคือการ get ข้อมูลของการ search ข้อมูลทั้ง user และทำการ return ข้อมูลทั้งหมดกลับไป
- 4) getCollectionsSearchService จะมีการทำงานคือการ get ข้อมูลของการ search ข้อมูลทั้ง collection และทำการ return ข้อมูลทั้งหมดกลับไป

ใน searchValidation นั้นจะมีการ validate ดังนี้

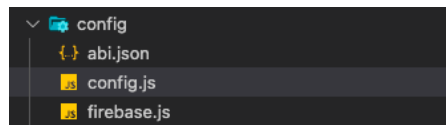
```
const getAllSearchValidate = {  
  query: joi.object().keys({  
    keyword: joi.string().required(),  
  }),  
};  
  
const getNFTsSearchValidate = {  
  query: joi.object().keys({  
    keyword: joi.string().required(),  
  }),  
};  
  
const getUsersSearchValidate = {  
  query: joi.object().keys({  
    keyword: joi.string().required(),  
  }),  
};  
  
const getCollectionsSearchValidate = {  
  query: joi.object().keys({  
    keyword: joi.string().required(),  
  }),  
};
```

รูปที่ 65 getAllSearchValidate

- 1) getAllSearchValidate ที่มีการสร้าง query จะต้องเป็น object ที่มี
- 2) keyword เป็น string
- 3) getNFTsSearchValidate ที่มีการสร้าง query จะต้องเป็น object ที่มี
- 4) keyword เป็น string
- 5) getUsersSearchValidate ที่มีการสร้าง query จะต้องเป็น object ที่มี
- 6) keyword เป็น string
- 7) getCollectionsSearchValidate ที่มีการสร้าง query จะต้องเป็น object ที่มี
- 8) keyword เป็น string

ทั้งหมดเป็น route ทั้งหมดที่มีอยู่ในระบบของ backend แต่ต่อไปจะอธิบายถึงส่วนที่มีการเรียกใช้เหมือนกันในระบบคือ ส่วนของ config, middlewares, utils

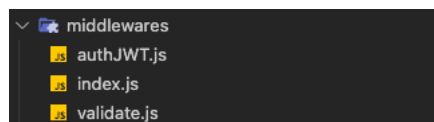
- 1) config



รูปที่ 66 getAllSearchValidate

Folder config จะมีดังนี้ไฟล์ abi.json ซึ่งจะมี event ของ สำหรับการดักอ่าน log ของ transactionHash และจะมีไฟล์ config.js ที่จะมีการกำหนดค่าต่างๆ ที่ใช้ในทางระบบ และมีไฟล์ firebase.js ที่จะต้อง config ไฟล์สำหรับการติดต่อกับระบบ firebase

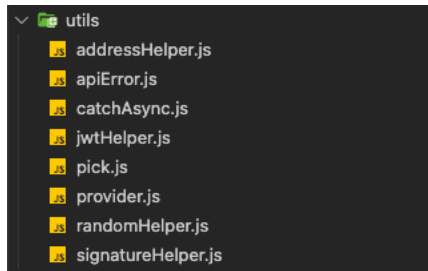
- 2) middlewares



รูปที่ 67 getAllSearchValidate

Folder middlewares จะมีไฟล์ดังนี้ authJWT.js ที่จะมีการ validate token ที่แนบมาในส่วน ของ header และไฟล์ validate จะมีการทำ validate ตัวของ params, query, body

3) utils



รูปที่ 68 utils

Folder utils จะมี

- 1) addressHelper ที่ใช้ในการ validate address ว่าเป็น address ของ crypto waller จริง หรือเปล่า

```
const isValidEthAddress = (address) => Web3.utils.isAddress(address);
```

รูปที่ 69 isValidEthAddress

- 2) catchAsync จะเป็นการทำ catch error หากเกิด error ขึ้น

```
const catchAsync = (fn) => (req, res, next) => {  
  Promise.resolve(fn(req, res, next)).catch((err) => next(err));  
};
```

รูปที่ 70 catchAsync

- 3) jwtHelper จะแบ่งออกเป็นดังนี้ jwtGenerate, jwtRefreshTokenGenerate, jwtVerifyToken

```
const jwtGenerate = (address) => {
  const accessToken = jwt.sign({ address }, config.jwtSecretKey, {
    expiresIn: "3m",
    algorithm: "HS256",
  });

  return accessToken;
};

const jwtRefreshTokenGenerate = (address) => {
  const refreshToken = jwt.sign({ address }, config.jwtSecretKey, {
    expiresIn: "1d",
    algorithm: "HS256",
  });

  return refreshToken;
};

const jwtVerifyToken = (token) => {
  const verifyToken = jwt.verify(token, config.jwtSecretKey, (err, decoded) => {
    if (err) throw new Error(error);
  });
  console.log(verifyToken);

  return verifyToken;
};
```

รูปที่ 71 jwtGenerate

โดยฟังก์ชัน jwtGenerate จะมีการรับ address และทำการ generate access token และทำการ return access token กลับไป ฟังก์ชัน jwtRefreshTokenGenerate จะมีการรับ address และทำการ generate access token และทำการ return access token กลับไป และสุดท้ายคือ jwtVerifyToken จะมีการ verify token และ return ผลลัพธ์กลับ

- 1) provider จะทำการ return provider กลับไป

```
const getProvider = () => {
  return new ethers.providers.JsonRpcProvider("https://rpc2.sepolia.org/ ");
};
```

รูปที่ 72 getProvider

- 2) randomHelper จะมีการรับ ความยาวของเลข และ จำนวนที่มากที่สุด และทำการ return list ของ number กลับมาแบบสุ่ม

```
const randomNumber = (numberLength, maximumNumber) => {
  let numberList = [];
  if (numberLength > maximumNumber) {
    for (let i = 0; i < maximumNumber; i++) {
      while (numberList.length === i) {
        let sameData = false;
        const numberRes = Math.floor(Math.random() * numberLength);
        for (let y = 0; y < numberList.length; y++) {
          if (numberRes === numberList[y]) {
            sameData = true;
          }
        }
        if (sameData === false) {
          numberList.push(numberRes);
        }
      }
    }
  } else {
    for (let i = 0; i < numberLength; i++) {
      numberList.push(i);
    }
  }
  // console.log(numberList);
  return numberList;
};
```

รูปที่ 73 randomNumber

- 3) signatureHelper จะมีกาารรับ address, signature, messageToSign เพื่อ signature และ address

```
const isValidSignature = (address, signature, messageToSign) => {
  if (!address || typeof address !== "string" || !signature || !messageToSign) {
    return false;
  }

  const signingAddress = recoverPersonalSignature({
    data: messageToSign,
    sig: signature,
  });

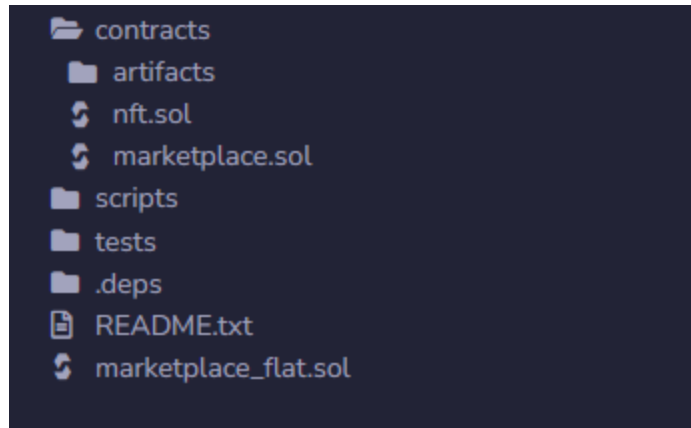
  if (!signingAddress || typeof signingAddress !== "string") {
    return false;
  }

  return signingAddress.toLowerCase() === address.toLowerCase();
};
```

รูปที่ 74 isValidSignature

3. Smart contract

smart contract โดยพัฒนาโดยใช้ remix ในการพัฒนาโดยเขียนเป็นภาษา solidity และ deploy ไว้ในระบบ etheruem sepolia testnet



รูปที่ 75 โครงสร้าง folder ของ smart contract

ซึ่งจะเห็นได้ว่าจะมี folder ดังภาพโดยจะเริ่มจากการเข้าไปที่ folder scripts



รูปที่ 76 folder scripts

จากภาพจะเห็น file deploy_ethers.js ซึ่งจะเป็นไฟล์สำหรับการ deploy smart contract ในระบบ blockchain

```

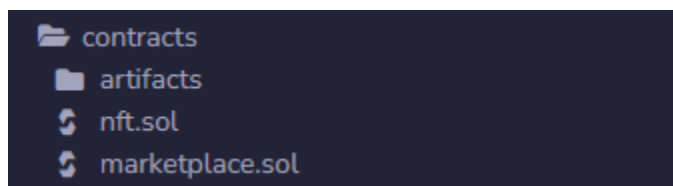
1 // Right click on the script name and hit "Run" to execute
2 (async () => {
3   try {
4     console.log('Running deployWithEthers script...')
5
6     const contractName = 'Storage' // Change this for other contract
7     const constructorArgs = [] // Put constructor args (if any) here for your contract
8
9     // Note that the script needs the ABI which is generated from the compilation artifact.
10    // Make sure contract is compiled and artifacts are generated
11    const artifactsPath = `browser/contracts/artifacts/${contractName}.json` // Change this for different path
12
13    const metadata = JSON.parse(await remix.call('fileManager', 'getFile', artifactsPath))
14    // 'web3Provider' is a remix global variable object
15    const signer = (new ethers.providers.Web3Provider(web3Provider)).getSigner()
16
17    let factory = new ethers.ContractFactory(metadata.abi, metadata.data.bytecode.object, signer);
18
19    let contract = await factory.deploy(...constructorArgs);
20
21    console.log('Contract Address: ', contract.address);
22
23    // The contract is NOT deployed yet; we must wait until it is mined
24    await contract.deployed()
25    console.log('Deployment successful.')
26  } catch (e) {
27    console.log(e.message)
28  }
29 })()

```

รูปที่ 77 script สำหรับการ deploy

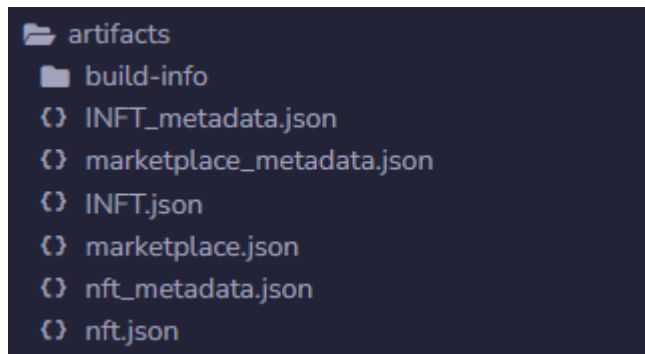
ในส่วนนี้จะมีการ deploy smart contract จาก path ของ artifact ตามที่กำหนดไว้ในบรรทัดที่ 11 จากการกำหนดชื่อของ contractName ในบรรทัดที่ 6 และจะมีการเรียกตัวของ metadata ตาม path ของ artifacts ที่ได้กำหนดไว้ในบรรทัดที่ 13 ในบรรทัดที่ 15 จะเป็นการ get ข้อมูลของผู้ที่ทำการ deploy smart contract และข้อมูลของ blockchain network จาก crypto wallet และทำการสร้าง contract factory และทำการ deploy smart contract

ส่วนต่อไปเป็นส่วนของ contract ที่มีการเรียกในการ deploy contract



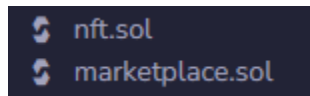
รูปที่ 78 folder ของ contracts

จากการเรียกในการ deploy จะไปที่ folder artifacts ที่อยู่ใน folder contracts ซึ่งจะมีไฟล์ json ของ smart contract หลังจากการ compile smart contract โดยที่ไฟล์ทั้งหมดจะเป็นไฟล์ abi กับ bytecode ของ smart contract ดังภาพ



รูปที่ 79 artifacts ของ contracts

ซึ่งข้อมูลใน artifacts ทั้งหมดนี้มาจาก 2 ไฟล์ smart contract ดังนี้



รูปที่ 80 file smart contract

1) NFT

nft contract นี้จะมี constructor คือ name และ symbol ของ contract ดังนี้

```
constructor(string memory name_, string memory symbol_) {  
    _name = name_;  
    _symbol = symbol_;  
}
```

รูปที่ 81 constructor ของ NFT contract

```

using Address for address;
using Strings for uint256;

using Counters for Counters.Counter;

Counters.Counter private _tokenIds;

string private _name;

string private _symbol;

struct NFT {
    address _owners;
    address[] _collaborators;
    uint256[] _collaboratorsPercentage;
    string _tokenURI;
    uint256 counterTranfers;
}

mapping(address => uint256) private _balances;

mapping(uint256 => address) private _tokenApprovals;

mapping(address => mapping(address => bool)) private _operatorApprovals;

mapping(uint256 => NFT) private tokenIdToNFT;

```

รูปที่ 82 การกำหนดตัวแปรใน NFT contract

จากภาพเป็นการกำหนดตัวแปรที่จะใช้เก็บข้อมูลลงใน blockchain และในไฟล์ของ nft contract นี้จะมีทั้งหมด 31 function, 1 event

```

event Mint (
    uint256 tokenId,
    address owner
);

```

รูปที่ 83 event สำหรับการ mint NFT

1 event คือ event ของการ mint โดยที่สร้างไว้สำหรับการดักจับ logs ของ event ของ transactionHash เพื่อใช้แสดงในส่วนของ history ของ NFT และ function ที่ใช้ใน nft contract นั้นจะมีทั้งหมดดังนี้

```

function _incrementCounterTransfers(uint256 tokenId) public {
    uint256 count = tokenIdToNFT[tokenId].counterTranfers;
    tokenIdToNFT[tokenId].counterTranfers = count+1;
}

```

รูปที่ 84 function _incrementCounterTransfers

- 1) `_incrementCounterTransfers` ใช้สำหรับการนับจำนวนการขายของ nft เพื่อใช้สำหรับการแบ่งผลประโยชน์ครั้งแรกและครั้งต่อไป โดยจะมีรับ input เป็น `tokenId` และไม่มี output

```
function getCounterTransfers(uint256 tokenId) public view returns (uint256) {
    uint256 count = tokenIdToNFT[tokenId].counterTranfers;
    return count;
}
```

รูปที่ 85 function `getCounterTransfers`

- 2) `getCounterTransfers` ใช้สำหรับการดูจำนวนการซื้อขายของ nft ซึ่งจะรับ input เป็น id ของ token ที่ต้องการและจะมีการ return จำนวนการซื้อขายของ nft

```
function tokenURI(uint256 tokenId) public view virtual override returns (string memory) {
    require(_exists(tokenId), "ERC721URIStorage: URI query for nonexistent token");

    string memory _tokenUri = tokenIdToNFT[tokenId]._tokenURI;
    string memory base = _baseURI();

    if (bytes(base).length == 0) {
        return _tokenUri;
    }
    if (bytes(_tokenUri).length > 0) {
        return string(abi.encodePacked(base, _tokenUri));
    }

    return _tokenURI(tokenId);
}
```

รูปที่ 86 function `tokenURI`

- 3) `tokenURI` ซึ่งจะรับ input เป็น id ของ token ที่ต้องการ และจะมีการ return ข้อมูล URI ของ nft

```
function mint(address[] memory collaborator,uint256[] memory collaboratorPercent,string memory uri) public {
    _tokenIds.increment();
    uint256 tokenId = _tokenIds.current();

    _safeMint(msg.sender, collaborator,collaboratorPercent,tokenId);
    _setTokenURI(tokenId, uri);
}
```

รูปที่ 87 function `mint`

- 4) mint ซึ่ง function นี้จะมีหน้าที่เป็นตัวกลางระหว่างการ set ข้อมูล uri และการสร้างข้อมูลอื่นๆ ของ nft ซึ่งมีการรับ input อยู่ 3 ตัวคือ array ของ address ของ collaborator, array ของ percent ของแต่ละ collaborator และ uri ของรูปภาพ และไม่มี output

```
function _safeMint(
    address to,
    address[] memory collaborator,
    uint256[] memory collaboratorPercent,
    uint256 tokenId,
    bytes memory data
) internal virtual {
    _mint(to, collaborator, collaboratorPercent, tokenId);
    require(
        _checkOnERC721Received(address(0), to, tokenId, data),
        "ERC721: transfer to non ERC721Receiver implementer"
    );
}
```

รูปที่ 88 function _safeMint

- 5) _safeMint โดย function นี้จะหน้าที่เป็นตัวกลางก่อนการ mint ซึ่งมีการรับ input สำหรับการ mint ดังนี้ to, collaborator, collaboratorPercent, tokenId เพื่อส่งต่อไปให้ function ของการสร้าง nft และ function นี้ไม่มีการ output

```
function _mint(address to, address[] memory collaborator, uint256[] memory collaboratorPercent, uint256 tokenId) internal virtual {
    require(to != address(0), "ERC721: mint to the zero address");
    require(!_exists(tokenId), "ERC721: token already minted");

    unchecked {
        _balances[to] += 1;
    }

    emit Mint(
        tokenId,
        to
    );

    tokenIdToNFT[tokenId]._owners = to;
    tokenIdToNFT[tokenId]._collaborators = collaborator;
    tokenIdToNFT[tokenId]._collaboratorsPercentage = collaboratorPercent;

    emit Transfer(address(0), to, tokenId);
}
```

รูปที่ 89 function _mint

- 6) `_mint` โดย function สำหรับการ `mint` นี้จะเป็นการสร้าง `nft` ซึ่งมีการรับ input คือ `to`, `collaborator`, `collaboratorPercent`, `tokenId` และ function นี้ไม่มี output

```
function _setTokenURI(uint256 tokenId, string memory _tokenUri) internal virtual {
    require(_exists(tokenId), "ERC721URIStorage: URI set of nonexistent token");
    tokenIdToNFT[tokenId]._tokenURI = _tokenUri;
}
```

รูปที่ 90 function `_setTokenURI`

- 7) `_setTokenURI` เป็น function สำหรับการ `set` ข้อมูลของ `token uri` โดยมีการรับ input คือ `id` ของ `token` และ `uri` ของรูปภาพ และไม่มี output

```
function balanceOf(address owner) public view virtual override returns (uint256) {
    require(owner != address(0), "ERC721: address zero is not a valid owner");
    return _balances[owner];
}
```

รูปที่ 91 function `balanceOf`

- 8) `balanceOf` เป็น function สำหรับการเช็คจำนวน `nft` ทั้งหมดที่เจ้าของ `address` เป็นเจ้าของอยู่ โดยที่จะมี input คือ `owner` และจะมี output คือจำนวน `nft` ทั้งหมดที่เป็นเจ้าของอยู่

```
function ownerOf(uint256 tokenId) public view virtual override returns (address) {
    address owner = tokenIdToNFT[tokenId]._owners;
    require(owner != address(0), "ERC721: invalid token ID");
    return owner;
}
```

รูปที่ 92 function `ownerOf`

- 9) `ownerOf` เป็น function สำหรับการเช็คเจ้าของ `token` โดยที่จะรับ Input เป็น `tokenId` และจะมี output คือเจ้าของ `nft` เป็น `address`

```
function getTokenCurrent() public view virtual returns (string memory) {
    return _tokenIds.current().toString();
}
```

รูปที่ 93 function getTokenCurrent

- 10) getTokenCurrent เป็น function สำหรับการเช็คจำนวน token ในปัจจุบันทั้งหมด โดยที่ไม่มี input แต่จะมี output เป็นจำนวนทั้งหมดของ nft

```
function collaboratorOf(uint256 tokenId) public view virtual returns (address[] memory) {
    address[] memory collaborator = tokenIdToNFT[tokenId]._collaborators;
    return collaborator;
}
```

รูปที่ 94 function collaboratorOf

- 11) collaboratorOf เป็น function สำหรับการเช็ค collaborator ของ token โดยที่จะรับ Input เป็น tokenId และจะมี output คือ address ของ collaborator

```
function collaboratorPercentageOf(uint256 tokenId) public view virtual returns (uint256[] memory) {
    uint256[] memory collaboratorPercentage = tokenIdToNFT[tokenId]._collaboratorsPercentage;
    return collaboratorPercentage;
}
```

รูปที่ 95 function collaboratorPercentOf

- 12) collaboratorPercentOf เป็น function สำหรับการเช็ค per ของ collaborator ของ token โดยที่จะรับ Input เป็น tokenId และจะมี output คือ จำนวน percent ของ collaborator

```
function name() public view virtual override returns (string memory) {
    return _name;
}
```

รูปที่ 96 function name

- 13) name เป็น function สำหรับเช็คชื่อของ nft contract ที่ได้ทำการ set ไว้ใน constructor ซึ่งไม่มี Input แต่จะ return ชื่อของ nft contract


```
function symbol() public view virtual override returns (string memory) {
    return _symbol;
}
```

รูปที่ 97 function symbol

- 14) symbol เป็น function สำหรับเช็ค symbol ของ nft contract ที่ได้ทำการ set ไว้ใน constructor ซึ่งไม่มี Input แต่จะ return symbol ของ nft contract

```
function _tokenURI(uint256 tokenId) private view returns (string memory) {
    require(_exists(tokenId), "ERC721Metadata: URI query for nonexistent token");

    string memory baseURI = _baseURI();
    return bytes(baseURI).length > 0 ? string(abi.encodePacked(baseURI, tokenId.toString())) : "";
}
```

รูปที่ 98 function _tokenURI

- 15) _tokenURI เป็น function สำหรับดู uri ของ nft ที่จะมีการเรียกจาก function tokenURI โดยที่จะมีการรับ input เป็น tokenId และมี output เป็น uri ของ tokenId return ไปให้ function tokenURI

```
function _baseURI() internal view virtual returns (string memory) {
    return "";
}
```

รูปที่ 99 function _baseURI

- 16) _baseURI เป็น function สำหรับ return string ว่ากลับไปให้ function ที่เรียกใช้งาน โดยที่ไม่มี input แต่มี output เป็น string ว่า

```
function approve(address to, uint256 tokenId) public virtual override {
    address owner = nft.ownerOf(tokenId);
    require(to != owner, "ERC721: approval to current owner");

    require(
        _msgSender() == owner || isApprovedForAll(owner, _msgSender()),
        "ERC721: approve caller is not token owner or approved for all"
    );

    _approve(to, tokenId);
}
```

รูปที่ 100 function approve

- 17) approve เป็น function สำหรับการเช็คต่างๆ เพื่อที่จะอนุมัติก่อนการโอน token โดยที่จะรับ Input เป็น to, tokenId และจะไม่มี output

```
function getApproved(uint256 tokenId) public view virtual override returns (address) {
    _requireMinted(tokenId);

    return _tokenApprovals[tokenId];
}
```

รูปที่ 101 function getApproved

- 18) getApproved เป็น function สำหรับ invalid ของ tokenId โดยที่จะมี input เป็น tokenId และจะมีการ และจะมี output เป็น address

```
function setApprovalForAll(address operator, bool approved) public virtual override {
    _setApprovalForAll(_msgSender(), operator, approved);
}
```

รูปที่ 102 function setApprovalForAll


- 19) setApprovalForAll เป็น function สำหรับเป็นตัวกลางสำหรับการเรียกใช้ _setApprovalForAll โดยที่จะมี input เป็น operator, approved แต่ไม่มี output

```
function isApprovedForAll(address owner, address operator) public view virtual override returns (bool) {
    return _operatorApprovals[owner][operator];
}
```

รูปที่ 103 function isApprovedForAll

- 20) isApprovedForAll เป็น function ที่ใช้สำหรับการเช็คการอนุมัติ โดยที่มีการรับ input เข้ามาเช็คคือ owner, operator และมี output เป็นผลการเช็คเป็น bool

```


function transferFrom(     infinite gas
    address from,
    address to,
    uint256 tokenId
) public virtual override {
    _transfer(from, to, tokenId);
}

```

รูปที่ 104 function transferFrom

- 21) transferForm เป็น function ที่เป็นตัวกลางสำหรับการเรียกใช้ transfer โดยที่จะมีการรับ input เพื่อที่จะส่งไปที่ function _transfer คือ from, to และ tokenId โดยที่ไม่มี output

```

function safeTransferFrom(     infinite gas
    address from,
    address to,
    uint256 tokenId
) public virtual override {
    safeTransferFrom(from, to, tokenId, "");
}

```

รูปที่ 105 function safeTransferForm

- 22) safeTransferForm เป็น function ที่เป็นตัวกลางสำหรับการเรียกใช้ safeTransferForm โดยที่จะมีการรับ input เพื่อที่จะส่งไปที่ function safeTransferForm คือ from, to และ tokenId โดยที่ไม่มี output

```
function safeTransferFrom(
    address from,
    address to,
    uint256 tokenId,
    bytes memory data
) public virtual override {
    require(_isApprovedOrOwner(msgSender(), tokenId), "ERC721: caller is not token owner or approved");
    _safeTransfer(from, to, tokenId, data);
}
```

รูปที่ 106 function safeTransferFrom

23) safeTransferFrom เป็น function ที่เป็นตัวกลางสำหรับการเรียกใช้ _safeTransfer ซึ่งจะมีการเช็ค require เจ้าของ token โดยที่จะมีการรับ input เพื่อที่จะส่งไปที่ function _safeTransfer คือ from, to, tokenId และ byte ของ data โดยที่ไม่มี output

```
function _safeTransfer(
    address from,
    address to,
    uint256 tokenId,
    bytes memory data
) internal virtual {
    _transfer(from, to, tokenId);
    require(_checkOnERC721Received(from, to, tokenId, data), "ERC721: transfer to non ERC721Receiver implementer");
}
```

รูปที่ 107 function _safeTransfer

24) _safeTransfer เป็น function ที่เป็นตัวกลางสำหรับการเรียกใช้ _transfer ซึ่งจะมีการเช็ค ERC721Received โดยที่จะมีการรับ input เพื่อที่จะส่งไปที่ function _safeTransfer คือ from, to, tokenId และ byte ของ data โดยที่ไม่มี output

```
function _exists(uint256 tokenId) internal view virtual returns (bool) {
    return tokenIdToNFT[tokenId]._owners != address(0);
}
```


รูปที่ 108 function _exists

25) _exists เป็น function ที่ใช้สำหรับการเช็ค address ของเจ้าของ nft owner ไม่เท่ากับ 0 โดยจะมีการรับ Input เป็น tokenId และมี output เป็น bool

```
function _isApprovedOrOwner(address spender, uint256 tokenId) internal view virtual returns (bool) {
    address owner = nft.ownerOf(tokenId);
    return (spender == owner || isApprovedForAll(owner, spender) || getApproved(tokenId) == spender);
}
```

รูปที่ 109 function _isApprovedOrOwner

- 26) `_isApprovedOrOwner` เป็น function ที่ใช้สำหรับการอนุมัติ address โดยที่จะมีการรับ input เป็น sender, tokenId และมี output เป็น bool

```
function _transfer(  infinite gas
    address from,
    address to,
    uint256 tokenId
) internal virtual {
    require(nft.ownerOf(tokenId) == from, "ERC721: transfer from incorrect owner");
    require(to != address(0), "ERC721: transfer to the zero address");

    require(nft.ownerOf(tokenId) == from, "ERC721: transfer from incorrect owner");

    // Clear approvals from the previous owner
    delete _tokenApprovals[tokenId];

    unchecked {
        _balances[from] -= 1;
        _balances[to] += 1;
    }
    _incrementCounterTransfers(tokenId);
    tokenIdToNFT[tokenId]._owners = to;

    emit Transfer(from, to, tokenId);
}
```

รูปที่ 110 function `_transfer`


- 27) `_transfer` เป็น function ที่ใช้สำหรับการเปลี่ยนเจ้าของ nft โดยที่จะมีการรับ input เป็น from, to, tokenId และไม่มี output

```
function _approve(address to, uint256 tokenId) internal virtual {
    _tokenApprovals[tokenId] = to;
    emit Approval(nft.ownerOf(tokenId), to, tokenId);
}
```

รูปที่ 111 function `_approve`

- 28) `_approve` เป็น function ที่ใช้สำหรับการ set approvals ของ token โดยที่จะรับ input คือ to, tokenId และไม่มี output

```

function _setApprovalForAll(  infinite gas
    address owner,
    address operator,
    bool approved
) internal virtual {
    require(owner != operator, "ERC721: approve to caller");
    _operatorApprovals[owner][operator] = approved;
    emit ApprovalForAll(owner, operator, approved);
}

```

รูปที่ 112 function _setApprovalForAll

- 29) _setApprovalForAll เป็น function ที่ใช้สำหรับการ set operatorApprovals โดยที่จะรับ input คือ owner, operator, approved และไม่มี output

```

function _requireMinted(uint256 tokenId) internal view virtual {
    require(_exists(tokenId), "ERC721: invalid token ID");
}

```

รูปที่ 113 function _requireMinted

- 30) _requireMinted เป็น function ที่ใช้สำหรับการเช็ค invalid ของ token Id โดยที่จะรับ input คือ tokenId และไม่มี output

```

function _checkOnERC721Received(  undefined gas
    address from,
    address to,
    uint256 tokenId,
    bytes memory data
) private returns (bool) {
    if (to.isContract()) {
        try IERC721Receiver(to).onERC721Received(_msgSender(), from, tokenId, data) returns (bytes4 retval) {
            return retval == IERC721Receiver.onERC721Received.selector;
        } catch (bytes memory reason) {
            if (reason.length == 0) {
                revert("ERC721: transfer to non ERC721Receiver implementer");
            } else {
                /// @solidity memory-safe-assembly
                assembly {
                    revert(add(32, reason), mload(reason))
                }
            }
        }
    }
    return true;
}

```

รูปที่ 114 function _checkOnERC721Received

31) `_checkOnERC721Received` เป็น function ที่ใช้สำหรับการเช็ค ERC 721 Received โดยที่จะรับ input คือ `from`, `to`, `tokenId`, `byte` ของ `data` และมี output เป็น `bool`

2. marketplace

marketplace contract นี้จะมี constructor คือ owner ของ contract ดังนี้

```
constructor() { 1835141 gas 1764800 gas
    owner = msg.sender;
}
```

รูปที่ 115 constructor ของ marketplace contract

```
using Counters for Counters.Counter;
using SafeMath for uint256;
Counters.Counter private _itemIds;
Counters.Counter private _itemsSold;

struct MarketItem {
    uint itemId;
    address nftContract;
    uint256 tokenId;
    address payable seller;
    address payable owner;
    uint256 price;
    bool sold;
}

address public owner;
uint256 listingPrice = 0.025 ether;

mapping(uint256 => MarketItem) private idToMarketItem;
```

รูปที่ 116 การกำหนดตัวแปรใน marketplace contract

จากภาพเป็นการกำหนดตัวแปรที่จะใช้เก็บข้อมูลลงใน blockchain และในไฟล์ของ marketplace contract นี้จะมีทั้งหมด 6 function, 3 event


```
event List (  
    uint indexed itemId,  
    address indexed nftContract,  
    uint256 indexed tokenId,  
    address seller,  
    address owner,  
    uint256 price,  
    bool sold  
);  
  
event Unlist (  
    uint indexed itemId,  
    address owner  
);  
  
event Sale (  
    uint indexed itemId,  
    address owner,  
    uint price  
);
```

รูปที่ 117 event ใน marketplace contract

3 event คือ event List, Unlist, Sale โดยที่สร้างไว้สำหรับการดักจับ logs ของ event ของ transactionHash เพื่อใช้แสดงในส่วนของ history ของ NFT

และ function ที่ใช้ใน marketplace contract นั้นจะมีทั้งหมดดังนี้


```

function listedNFTItem(  infinite gas
    address nftContract,
    uint256 tokenId,
    uint256 price
) public payable nonReentrant {
    require(price > 0, "Price must be greater than 0");

    _itemIds.increment();
    uint256 itemId = _itemIds.current();

    idToMarketItem[itemId] = MarketItem(
        itemId,
        nftContract,
        tokenId,
        payable(msg.sender),
        payable(address(0)),
        price,
        false
    );

    emit List(
        itemId,
        nftContract,
        tokenId,
        msg.sender,
        address(0),
        price,
        false
    );

    IERC721(nftContract).transferFrom(msg.sender, address(this), tokenId);
}

```

รูปที่ 118 function listedNFTItem

- 1) listedNFTItem เป็น function ที่ใช้สำหรับ nft ที่เราทำการสร้างเสร็จแล้วมาทำการลงขายใน marketplace โดยที่ input คือ nftContract, tokenId, price และไม่มี output

```

function itemFromTokenId(uint256 tokenId) public view returns (uint256) {
    uint itemCount = _itemIds.current();
    uint currentIndex = 0;

    MarketItem memory items;
    for (uint i = 0; i < itemCount; i++) {
        if (idToMarketItem[i + 1].tokenId == tokenId) {
            uint currentId = i + 1;
            MarketItem storage currentItem = idToMarketItem[currentId];
            items = currentItem;
            currentIndex += 1;
        }
    }
    return items.itemId;
}

```

รูปที่ 119 function itemFromTokenId

- 2) itemFromTokenId เป็น function ที่ใช้สำหรับการเช็ค itemId ของ token โดยที่จะ input ดังนี้คือ tokenId และมี output คือ itemId ของ token

```

function priceFromTokenId(uint256 tokenId) public view returns (uint256) {
    uint itemCount = _itemIds.current();
    uint currentIndex = 0;


    MarketItem memory items;
    for (uint i = 0; i < itemCount; i++) {
        if (idToMarketItem[i + 1].tokenId == tokenId) {
            uint currentId = i + 1;
            MarketItem storage currentItem = idToMarketItem[currentId];
            items = currentItem;
            currentIndex += 1;
        }
    }
    return items.price;
}

```

รูปที่ 120 function priceFromTokenId

- 3) priceFromTokenId เป็น function ที่ใช้สำหรับการเช็ค price ของ token โดยที่จะ input ดังนี้คือ tokenId และมี output คือ price ของ token

```


function saleNFTItem(  infinite gas
    address nftContract,
    uint256 itemId
) public payable {
    uint tokenId = idToMarketItem[itemId].tokenId;
    address seller = idToMarketItem[itemId].seller;
    uint256 price = idToMarketItem[itemId].price;
    require(msg.value == price, "Value must be equal price");
    emit Sale(
        itemId,
        msg.sender,
        price
    );
    address[] memory itemCollaborator = INFT(nftContract).collaboratorOf(tokenId);
    uint256[] memory itemCollaboratorPercent = INFT(nftContract).collaboratorPercentageOf(tokenId);
    uint256 count = INFT(nftContract).getCounterTransfers(tokenId);
    if(count > 1){
        if(itemCollaborator.length == 1){
            uint256 sumPercentOfSeller = 100 - itemCollaboratorPercent[0];
            payable(seller).transfer(msg.value * sumPercentOfSeller/100);
            payable(itemCollaborator[0]).transfer(msg.value * itemCollaboratorPercent[0]/100);
        }else{
            uint256 sumCollabPercent = itemCollaboratorPercent[0]+itemCollaboratorPercent[1];
            uint256 sumPercentOfSeller = 100 - sumCollabPercent;
            payable(seller).transfer(msg.value * sumPercentOfSeller/100);
            payable(itemCollaborator[0]).transfer(msg.value * itemCollaboratorPercent[0]/100);
            payable(itemCollaborator[1]).transfer(msg.value * itemCollaboratorPercent[1]/100);
        }
    }else{
        if(itemCollaborator.length == 1){
            payable(itemCollaborator[0]).transfer(msg.value);
        }else{
            uint256 sumCollabPercent = itemCollaboratorPercent[0]+itemCollaboratorPercent[1];
            payable(itemCollaborator[0]).transfer(msg.value * itemCollaboratorPercent[0]/sumCollabPercent);
            payable(itemCollaborator[1]).transfer(msg.value * itemCollaboratorPercent[1]/sumCollabPercent);
        }
    }
    IERC721(nftContract).transferFrom(address(this), msg.sender, tokenId);
    idToMarketItem[itemId].owner = payable(msg.sender);
    _itemsSold.increment();
    idToMarketItem[itemId].sold = true;
}

```

รูปที่ 121 function saleNFTItem

- 4) saleNFTItem เป็น function ที่ใช้สำหรับการขาย nft โดยที่จะ input ดังนี้คือ nftContract, itemId และ ไม่มี output

```

function unlistNFTItem(  infinite gas
address nftContract,
uint256 itemId
) public payable nonReentrant {
    uint tokenId = idToMarketItem[itemId].tokenId;
    bool sold = idToMarketItem[itemId].sold;
    address ownerNFT = idToMarketItem[itemId].seller;
    // address seller = idToMarketItem[tokenId].seller;
    require(sold != true, "This Sale has already finished");
    require(ownerNFT == msg.sender, "Only seller may unlist an item");

    emit Unlist(
        itemId,
        msg.sender
    );

    IERC721(nftContract).transferFrom(address(this), ownerNFT, tokenId);
    idToMarketItem[itemId].owner = payable(msg.sender);
    _itemsSold.increment();
    idToMarketItem[itemId].sold = true;
}

```

รูปที่ 122 function unListNFTItem

- 5) unListNFTItem เป็น function ที่ใช้สำหรับการยกเลิกการวางขาย nft ในระบบโดยที่จะมี input ดังนี้คือ nftContract, itemId และไม่มี output

```

function fetchNFTItems() public view returns (MarketItem[] memory) {
    uint itemCount = _itemIds.current();
    uint unsoldItemCount = _itemIds.current() - _itemsSold.current();
    uint currentIndex = 0;

    MarketItem[] memory items = new MarketItem[](unsoldItemCount);
    for (uint i = 0; i < itemCount; i++) {
        if (idToMarketItem[i + 1].owner == address(0)) {
            uint currentId = i + 1;
            MarketItem storage currentItem = idToMarketItem[currentId];
            items[currentIndex] = currentItem;
            currentIndex += 1;
        }
    }
    return items;
}

```

รูปที่ 123 function fetchNFTItem

- 6) `fetchNFTItem` เป็น function การดู nft ทั้งหมดที่ทำการลงขายอยู่ในระบบ โดยที่จะไม่มี input แต่มี output คือ array ของ market item