МІНІСТЕРСТВО ОСВІТИ НАУКИ УКРАЇНИ
СТРУКТУРНИЙ ПІДРОЗДІЛ «ФАХОВИЙ КОЛЕДЖ ЕКОНОМІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ПРАТ «ПВНЗ «ЗІЕІТ»

Кафедра економічної кібернетики та інженерії програмного забезпечення

ДО ЗАХИСТУ ДОПУЩЕНИЙ

Голова _____

Левицький С. І.

ВИПУСКНА РОБОТА МОЛОШОГО СПЕЦІАЛІСТА
ТЕМА: «РОЗРОБКА ВЕБДОДАТКУ ДЛЯ ГЕНЕРАЦІЇ ГРАФІЧНОЇ КОЛЕКЦІЇ ІЗ НАЯВНИХ ЗОБРАЖЕНЬ КОРИСТУВАЧА»

Виконав
Студент групи ІПЗ-118к9 _____ Головченко А.Ю.

Науковий керівник
Ст. викладач _____ Дереза Е.В.

Запоріжжя
2022

ЗАВДАННЯ:

1. Тема роботи: Розробка вебдодатку для генерації графічної колекції із наявних зображень користувача, затверджена рішенням кафедри (протокол № __ від _____)

2. Термін подання студентом закінченої роботи на кафедру:_____

3. Вихідні дані до роботи (проекту) (визначаються кількісні або (та) якісні показники, яким повинен відповідати об’єкт розробки): Розробка вебдодатку для генерації графічної колекції із наявних зображень користувача на мові TypeScript з використанням бібліотеки React

4. Вимоги до змісту (перелік питань, що їх належить розробити) (визначаються назви розділів або (та) перелік питань, які повинні увійти до тексту роботи) :

4.1 Розгляд предметної області та огляд існуючих аналогів;

4.2 Вибір та обґрунтування програмних засобів та технологій для реалізації проекту;

4.3 Реалізація проекту з використанням описаних методик;

5. Індивідуальний план виконання

№ етапу	Зміст	Термін виконання
1	Формування теми курсового проекту	27.04.2021
2	Збір практичного матеріалу за темою	04.05.2021
3	Робота над створенням курсового проекту	25.05.2021
4	Оформлення пояснювальної записки та докладу	02.06.2021
5	Захист курсового проекту	12.06.2021

Керівник (ПІБ, підпис) _____

З планом ознайомлений (ПІБ, підпис) _____

РЕФЕРАТ

Пояснювальна записка складається з: сторінок – 76, рисунків – 22, лістинг коду – 22.

Об’єкт дослідження – вебдодаток для генерацій графічних колекцій.

Мета роботи – розробка вебдодатку для генерації графічної колекції із наявних зображень користувача на мові TypeScript з використанням бібліотеки React.

Методи дослідження – використання мови програмування JavaScript, використання мови програмування TypeScript, бібліотеки React, Visual Studio Code.

Розглянуто опис поточного стану в області генерації NFT колекцій. Зокрема розглянуті існуючі аналоги вебдодатків для генерації графічних колекцій та окремих послуг щодо генерації графічних колекцій програмним шляхом в індивідуальному порядку.

Обґрунтовано вибір середовища розробки та технологій що використовувалися для розробки вебдодатку.

В ході розробки були виконані наступні кроки:

- розроблено інтерфейс та функціональну частину вебдодатку;
- розроблено алгоритм генерації зображень вебдодатку;
- суміщено алгоритм та візуальна частина вебдодатку;
- протестований функціонал вебдодатку;
- оглянуто додаток та зроблені висновки.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

Слово/словосполучення	Скорочення	Умова використання
A		
Application Programming Interface	API	В тексті
American Standard Code for Information Interchange	ASCII	В тексті
C		
Cascading Style Sheets	CSS	В тексті
D		
Denial-of-service attack	DDoS	В тексті
Document Object Model	DOM	В тексті
E		
ECMAScript 6	ES6	В тексті
H		
HyperText Markup Language	HTML	В тексті
J		
JavaScript	JS	В тексті
JavaScript Object Notation	JSON	В тексті
N		
Node Package Manager	NPM	В тексті
Non-fungible token	NFT	В тексті

ВСТУП

Сфера NFT та повязанних із цим технологій почала набувати популярності наприкінці 2020 року та досягнула свого піку наприкінці 2021 року, про що свідчить статистика популярності запиту «NFT» у пошуковій системі Google, він став найшуканішим у пошуковій системі запитом.

NFT представляє собою цифровий медіа контент що зберігається у певній мережі, у мережі люди мають можливість передавати NFT від одного крипто-гаманця до іншого, або за допомогою смарт-контрактів організовувати торгівлю такими цифровими активами за криптовалюту.

Найпопулярнішим видом медіа контенту яким торгують у вигляді NFT є цифрові малюнки, вони як і реальні малюнки мають автора, людину яка створила цю NFT, цю інформацію у мережі неможливо підробити тому вона надає NFT від відомих художників чи брендів додаткову вартість і культурну цінність. Такий медіа контент звісно містить у собі культурну цінність на рівні з реальними картинами, але не пояснює велику увагу до сфери від мільйонів людей по всьому світу. Таку увагу до неї можуть пояснити гроші, пік ажіотажу у сфепрі прийшовся на три місяці після серії реакордних продажів NFT, наприклад як коли картина «Everydays: the First 5000 Days» цифрового художника Майка Вінкелмана більш відомого під псевдонімом «Beeple» була продана з аукціону за 69.3 мільйони долларів, це пояснюється відносною відомістю художника і тим фактом що це NFT є коллажем 5000 картин які художник малював протягом 5000 днів, але не у кожного є великі гроші або 13 з половиною років на створення подібного NFT, а попит тоді був набагато більше наявної пропозиції, це породило сферу згенерованих зображень. Генерація зображеня допомогла усім охочим купити і усім охочим створити власні NFT. Генерація можлива завдяки програмному коду який економить художникам час, наприклад замість того щоб малювати 10 000 зображень, ходожнику достатньо умовно поділити зображення на шари та намалювати по декілька варіацій кожного шару, таким чином поділивши свою картину на 5 шарів та намалювавши до кожного шару 10 варіацій, художник за допомогою

програмного коду може згенерувати до 100 000 унікальних комбінацій цих зображень, саме це допомогає пришвидшити роботу художника та не залишити потенційних покупців без товару. Товар можуть купляти як і у цілях колекціонування так і з метою подальшого перепродажу за більшою ціною.

Не зважаючи на велику популярність існує лише невелика кількість додатків, що виконують подібні задачі, а перші подібні вебдодатки стали з’являтися лише на прикінці 2021 – початку 2022 років.

Велика популярність та актуальність теми визначає тему дипломної роботи – «Розробка вебдодатку для генерації графічної колекції із наявних зображень користувача».

Метою роботи буде створення зручного та швидкого вебдодатку для генерації графічних колекцій із наявних зображень користувача.

РОЗДІЛ 1
ОГЛЯД ПРЕДМЕТНОЇ ЧАСТИНИ

1.1 Технологія блокчейну на прикладі мережі Ethereum

Для початку необхідно розглянути складові основи існування NFT – блокчейн та розумні контракти.

Блокчейн (з англійської Blockchain – «ланцюг блоків») – це вид розподілених систем, який додає та зберігає данні у вигляді блоків, пов’язаних між собою за допомогою криптографії, при цьому кожен наступній блок містить у собі криптографічний хеш попереднього блока, мітку часу і дані про транзакцію, завдяки чому підробка даних у такій мережі стає теоретично неможливою [1].

Вперше технологія набула практичного застосування у 2009 році, коли була створена перша крипто валюта Bitcoin. У 2015 році з реалізацією платформи Ethereum технологія набула значного покращення, були запроваджені розумні контракти та захист від DDoS-атак (розподілена атака з метою зробити комп'ютерні ресурси недоступними користувачам за допомогою надсилання надмірної кількості запитів), на мережу шляхом введення плати за кожен запит до платформи. Платформа Ethereum призначена для створення децентралізованих онлайн-сервісів на базі технології блокчейн та розумних контрактів.

Розумні контракти — це програми, що зберігаються в блокчейні, які запускаються при дотриманні заздалегідь визначених умов. Зазвичай вони використовуються для автоматизації виконання угоди, щоб усі учасники могли бути негайно впевнені в результаті без участі посередника чи втрати часу. Вони також можуть автоматизувати робочий процес, запускаючи наступну дію, коли виконуються умови. Для опису правил розумних контрактів для платформи Ethereum була розроблена нова мова програмування Soldity [2].

1.2 NFT та розумні контракти NFT

NFT (з англійської абревіатура non-fungible tokens – «невзаємозамінні токени») – це токен, що зберігається в блокчейні у формі розподіленого каталогу та є невзаємозамінним. Право власності на NFT записується в блокчейні і може бути передане власником, що дозволяє торгувати такими токенами. NFT зазвичай містять посилання на цифрові файли, такі як фотографії, відео та аудіо. Оскільки такі токени можна однозначно ідентифікувати вони є невзаємозамінними [3][4].

Ethereum є найпопулярнішою мережею для торгівлі NFT, а цифрові зображення є найпопулярнішим NFT товаром, зважаючи на попит на такі товари, NFT цифрових зображень створюються щоденно, найрозповсюдженішими зараз є так звані генеративні зображення (зображення створені шляхом програмної генерації з наданих художником частин), вони зазвичай є чистиною генерованих колекцій, кількість зображень у такій колекції може бути будь якою, але найпопулярніші NFT колекції містять у собі від 5 до 10 тисяч зображень, що в свою чергу задовольняє великий первинний попит на колекцію серед користувачів та водночас зберігає статус її унікальності за рахунок в міру обмеженої кількості екземплярів. Куплю-продаж NFT уможливлюють розумні контракти.

Розумні контракти NFT — це механізм реалізації угоди купівлі-продажу між власником NFT і покупцем. Вони використовуються для автоматизації виконання угоди, щоб усі учасники могли бути негайно впевнені в результаті без участі посередника чи втрати часу [5].

Причини купівлі NFT різняться в залежності від товару, наприклад, NFT відомих художників купляють задля колекціонування чи для подальшого перепродажу та заробітку грошей, деякі ж NFT мають у собі й іншу цінність, наприклад надання права власності на реально існуючі предмети, наприклад після купівлі NFT під назвою «The Doge Crown» від відомого бренду Dolce & Gabbana, перший власник мав змогу отримати фізичну версію зображеного у

медіа файлі ювелірного виробу від цього бренду. Наявність же будь якого NFT з колекції «Bored Ape Yacht Club» є свідостством членства закритого клубу та перепусткою на приватні вечірки, острови та яхти клубу безкоштовно. Також NFT не обійшли стороною ігрову та музичну індустрії, створюється все більше ігор де кожен внутрішньо ігровий предмет чи персонаж є NFT та може бути вільно проданий іншим ігрокам, музиканти в свою чергу можуть отримувати гроші від продажу альбомів у вигляді NFT незалежно від музичних платформ чи лейблів. NFT що мають у собі ще щось крім цих самих зображень користуються великим інтересом і попитом, а їх ціни можуть значно перевищувати початково встановлені власником.

Зазвичай NFT розміщують на продаж та купляють на спеціальних платформах, які за допомогою наявного функціоналу допомагають користувачам визначити чи належать певний NFT до колекції, що їх цікавить, чи користується певна NFT колекція попитом, тощо. Продаж NFT які користуються великим попитом зазвичай здійснюється через аукціони на цих платформах. Прикладами перевірених та популярніших платформ з куплі-продажу NFT є OpenSea, Foundation, Rarible.

Розумні контракти та технологія блокчейну унеможливлюють підробку транзакції або NFT товару на рівні мережі. Не дивлячись на це, шахраї у сфері продажу NFT не рідкість і торгівля на не перевірених платформах або в особтсьлму форматі з неперевіреними людьми може призвести до втрати грошей або цифрового активу. Слід зауважити що NFT створений на основі певного медіа файлу являє собою посилання на цей медіа файл що зберігається у мережі, а не, наприклад, унікальне кодування двійкового коду цього медіа файлу, тому на основі одного медіа файлу можна створити безліч різних з боку мережі але однакових за медіа змістом NFT, які буде важко відрізнити один від одного для недосвідченого користувача, тому важливо купляти та продавати NFT лише на перевірених платформах, де надається доступ до всієї історії транзакцій за конкретним NFT та інформацію про його належність до тієї чи іншої NFT колекції.

Найпоширенішими та в водночас найпопулярнішими NFT колекціями є генеративні колекції, такі колекції містять у собі зображення що створені за допомогою алгоритму генерації зображень із наявних частин які окремо малюються художником. Наприклад намалюючи 20 різних фонів, 20 різних дерев та 20 різних хмар для картини, за допомогою генерації можна отримати 8000 унікальних зображень з хмарою деревом та фоном, що значно прискорює роботу художника.

1.3 Сучасні додатки для генерації зображень з метою створення NFT колекцій

Згенерувати свою колекцію цифрових зображень з наявних у користувача елементів наразі можливо двома способами:

- генерація алгоритмом створеним під певний проект та його особливості;
- генерація за допомогою відповідних вебдодатків або десктопних додатків які містять певний алгоритм генерації.

Генерація алгоритмом створеним під певний проект та його особливості надає максимальну якість генерації та широкий спектр опцій, таких як вірогідність додавання того чи іншого елементу до зображення, анімовані елементи зображення, тощо , але має високу вартість та відсутню універсальність для кінцевого користувача у зв’язку з розробкою алгоритму під певний проект та його користувача. Графічний інтерфейс у таких індивідуальних додатках зазвичай відсутній, вхідні зображення завантажуються безпосередньо у папку проекту, через яку алгоритм з ними і працює. Пропонуючи розробку такого алгоритму виконавці просять в середньому від 1 000 до 2 000 доларів США за розробку алгоритму та генерацію будь якої кількості зображень у залежності від потреби віддавати алгоритм «під ключ». Також слід зауважити що існують і безкоштовні версії таких додатків, для роботи з якими потрібні знання у програмуванні для налаштування та

редагування алгоритму під свій проект безпосередньо через середу розробки, звісно це не рівень базового користувача але ця альтернатива доступна будь кому у відкритому доступі.

Генерація за допомогою вебдодатків або десктопних додатків надає максимальну універсальність та значну кількість опцій, переважно якість генерованих зображень нижча у порівнянні з індивідуальним алгоритмом що розробляється під проект, все залежить від того наскільки вхідні зображення «підходять» під реалізований у додатку алгоритм генерації. Зазвичай ціни у таких додатках залежать від типу розповсюдження додатку, загалом за одну генеровану колекцію з 10 000 зображень з користувача візьмуть від 100 до 500 доларів.

Наразі тема генерації колекцій зображень набуває популярності, а перші повноцінні вебдодатки та дескопні додатки були створені лише наприкінці 2021 року і мають ряд вищевказаних недоліків.

Слід відзначити популярні та набираючи популярність вебдодатки та десктопні додатки для генерації колекцій зображень:

- вебдодаток NFT-Inator (https://nft-inator.com/)
- вебдодаток NFT Art Generator (https://nft-generator.art/)
- вебдодаток NFT Creator (https://designcloud.appypie.com/nft-creator)
- десктопний додаток NFT Generator Software (https://zootchain.com/)

Розглянемо більш детально декілька з них.

Вебдодаток NFT-Inator має комплексний функціональний інтерфейс у налаштуваннях присутні розширені опції рендеру, такі як розширення та формат вихідних зображень, присутній функціонал генерації вихідних зображень «в-ручну» який виконано у вигляді конструктору, генерація відбувається шляхом накладання зображень користувача шарами, так званим «шаровим» методом, у встановленій послідовності, що є стандартним і найпоширенішим принципом генерації вихідних зображень із наявних. Із особливостей – можливість створення NFT із кожного зображення сгенерованої колекції зображень у будь якій мережі на вибір та об’єднання у NFT колекцію,

також веб додаток надає «демо-режим» де надані готові елементи зображень для генерації тестової колекції. Із недоліків слід виділити непостійність генерації через критичні помилки, успішна генерація тестової колекції відбулася лише з третьої спроби.

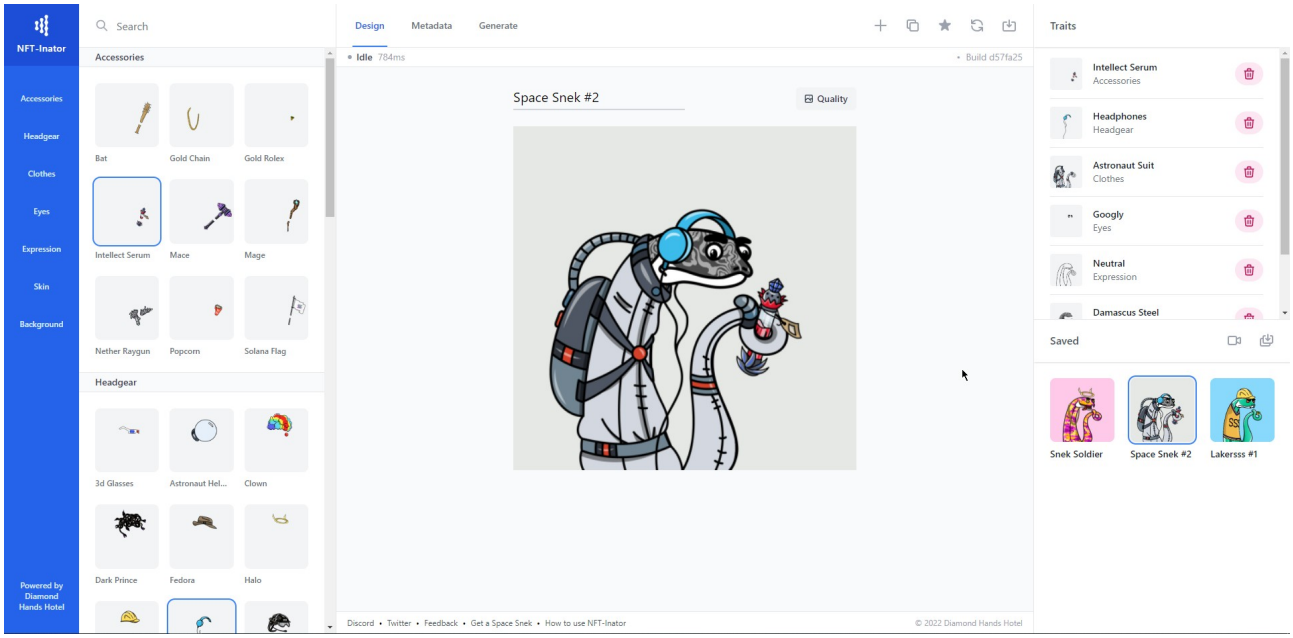

Рисунок 1.1 - Вигляд вебдодатку NFT-Inator.

Вебдодаток NFT Art Generator має зрозумілий інтерфейс, розширені налаштування алгоритму генерації з можливістю коригування шансів додавання того чи іншого типу елементу до вихідного зображення та опцію перев'ю для оцінки роботи алгоритму генерації, генерація відбувається стандартним «шаровим» методом. Із унікальних особливостей – має функцію роботи з анімованими зображеннями формату .gif, що дозволяє згенерувати колекцію у якій можуть бути присутні частково, або повністю анімовані зображення. Анімовані зображення є другими за популярністю медіа файлами на основі яких створюють NFT, а можливість генерації анімованої колекції зображень може стати вагомим аргументом при виборі користувачем додатку для генерації його колекції.

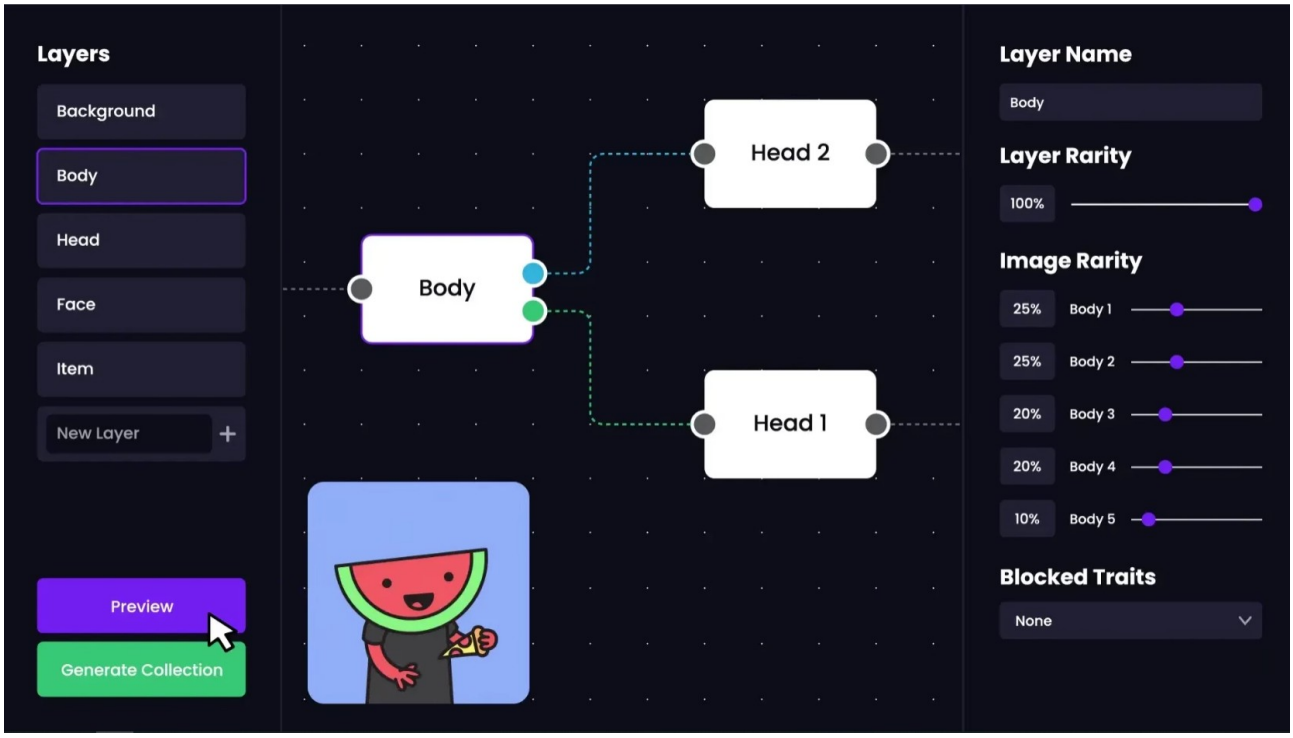

Рисунок 1.2 - Вигляд вебдодатку NFT Art Generator.

Оцінити функціонал десктопного додатку NFT Generator Software виявилося неможливо через його розповсюдження лише на оплатній основі. За існуючими промо відео з сайту постачальника неможливо визначити переваги та недоліки додатку.

Усі додатки мають функціонал завантаження зображень на сайт з яких буде відбуватися генерація колекції, ці зображення не зберігаються на сервері вебдодатку через можливі проблеми з авторськими правами, за тієї ж самої причини не передбачується збереження вихідних зображень на сервері вебдодатку.

Кінцевим результатом роботи таких додатків зазвичай є архів вихідних зображень, рідше, посилання на колекцію зображень у мережі блокчейну, останнє можливе за наявністю функціоналу завантаження вихідних зображень до блокчейну у вебдодатку та за умови сплати сплати «gas fee» (комісії мережі за запит у ній) користувачем. У мережі Ethereum «gas fee» наразі досягає 110 доларів США, а якщо взяти до уваги що кожен елемент колекції розміщується у мережі за окремим запитом, з такою комісією створення повної колекції є дуже

затратним процессом. Саме через високу вартість пріоритетним є надання користувачеві вихідних зображень напряму у вигляді архіву.

Найпопулярнішим методом для створення NFT є так званий «Mint» - перепродаж ще не завантаженої колекції, місця у якому визначаються за принципом лотереї, та надання вигравшим покупцям можливості купити отриманий ними медіа контент за привабливою ціною та оплатити «gas fee» тим самим створивши NFT у мережі та ставши його повноправним власником.

1.4 Постановка задачі

Для виконання випускної дипломної роботи на тему розробка вебдодатку для генерації графічної колекції із наявних зображень користувача необхідно виконати наступні задачі:

- аналіз та порівняння існуючих аналогів;
- аналіз та підбір інструментів для розробки програмної частини;
- розробка універсального алгоритму генерації колекції продукту;
- розробка функціоналу завантаження користувацьких зображень та створення архіву з результатом роботи алгоритму генерації – колекцією вихідних зображень;
- розробка інтерфейсу;
- тестування програмного продукту;
- написання пояснювальної записки дипломної роботи;
- написання висновків як для всієї робота, так і для кожного розділу;
- формування додатків до пояснювальної записки;
- підготовка презентації до захисту.

1.5 Висновки за розділом

Проаналізована актуальність сфери NFT та генеративних колекцій. Проаналізовані існуючі аналоги додатків для генерації колекцій зображень,

виділені їх особливості та недоліки. Описано структуру технічного завдання щодо роботи програмного продукту.

РОЗДІЛ 2
ОГЛЯД ПРОГРАМНОГО РІШЕННЯ

2.1 Мова програмування JavaScript

JavaScript — це інтерпретована, об'єктно-орієнтована мова з першокласними функціями, найбільш відома як мова сценаріїв для веб-сторінок, але вона також використовується в багатьох середовищах, що не є браузерами. Це заснована на прототипах, мультипарадигмальна мова сценаріїв, яка є динамічною та підтримує об’єктно-орієнтований, імперативний та функціональний стилі програмування. JavaScript працює на стороні клієнта, що можна використовувати для розробки/програмування поведінки веб-сторінок під час виникнення події [6].

Всупереч поширеній помилці, JavaScript не є «інтерпретованою Java». JavaScript – це динамічна мова сценаріїв, яка підтримує конструювання об’єктів на основі прототипів. Основний синтаксис навмисно схожий як на Java, так і на C++, щоб зменшити кількість нових понять, необхідних для вивчення мови. Мовні конструкції, такі як оператори if, цикли for і while, а також блоки switch і try ... catch функціонують так само, як і в цих мовах. JavaScript може функціонувати як процедурна, так і як об'єктно-орієнтована мова. Об’єкти створюються програмно в JavaScript, шляхом приєднання методів і властивостей до порожніх об’єктів під час виконання, на відміну від визначень синтаксичних класів, поширених у компільованих мовах, таких як C++ і Java. Після створення об’єкта його можна використовувати як план (або прототип) для створення подібних об’єктів.

Динамічні можливості JavaScript включають побудову об’єктів під час виконання, списки змінних параметрів, змінні функцій, створення динамічного сценарію (через eval), інтроспекцію об’єкта (через for ... in) та відновлення вихідного коду (програми JavaScript можуть декомпілювати тіла функцій назад у вихідний текст).

Синтаксис JS не відповідає потребам усіх веб розробників. Це і слід очікувати, тому що проекти та вимоги у всіх різні. Тому з’явилося безліч нових мов, які транспілюються (конвертуються) у JavaScript перед запуском у браузері.

Сучасні інструменти роблять транспіляцію дуже швидкою та прозорою, фактично дозволяючи розробникам кодувати іншою мовою та автоматично конвертувати її «під капотом». Приклади таких мов [7]:

- CoffeeScript — це «синтаксичний цукор» для JavaScript. Він вводить коротший синтаксис, що дозволяє нам писати більш чіткий і точний код. Зазвичай це подобається розробникам Ruby;
- TypeScript зосереджений на додаванні «суворої типізації» для спрощення розробки та підтримки складних систем. Його розробляє Microsoft;
- Flow також додає введення даних, але іншим способом. Розроблено Facebook;
- Dart — це окрема мова, яка має власний механізм, який працює в середовищі без браузера (наприклад, у мобільних додатках), але також може бути перенесений на JavaScript. Розроблено Google;
- Brython — це транспілятор Python на JS, який дозволяє писати програми на чистому Python без JS;
- Kotlin — сучасна, лаконічна та безпечна мова програмування, яка може орієнтуватися на браузер або Node. Є більше. Звичайно, навіть якщо ми використовуємо одну з транспілованих мов, ми також повинні знати JavaScript, щоб дійсно розуміти, що ми робимо.

2.2 Мова програмування TypeScript

TypeScript - це мова програмування, що компілюється в JS і є об'єктно-орієнтованою мовою програмування. Усі коди в TypeScript в кінцевому підсумку компілюються в JavaScript.

У TypeScript є кілька покращень, які дають йому перевагу над JavaScript. Ось список переваг TypeScript перед JavaScript [8]:

- Тільки під час розробки TypeScript виявляє проблеми компіляції. Це зменшує ймовірність помилок під час виконання;
- Ключова особлиість TypeScript полягає в тому, що він дозволяє статичну типізацію;
- Статична типізація дозволяє перевірити правильність типу під час компіляції. У JavaScript це неможливо;
- TypeScript — це не що інше, як JavaScript і деякі додаткові функції, наприклад функції ES6. Деякі з цих особливостей:
    - інтерфейси генерики;
    - простіри імен;
    - нульова перевірка;
    - модифікатори доступу.
- TypeScript підтримує IntelliSense, який надає активні підказки під час додавання коду.

2.3 Бібліотека ReactJS, Redux та Redux Saga

ReactJS — це декларативна, ефективна та гнучка бібліотека JavaScript для створення компонентів інтерфейсу користувача, які можна повторно використовувати. Це відкрита, компонентна, передня бібліотека, відповідальна лише за рівень перегляду програми. Його створив Джордан Волк, який був інженером-програмістом у Facebook. Спочатку він був розроблений і

підтримуваний Facebook, а потім був використаний у таких продуктах, як WhatsApp та Instagram. Facebook розробив ReactJS у 2011 році в розділі стрічки новин, але він був опублікований у травні 2013 року [9].

Додаток ReactJS складається з кількох компонентів, кожен з яких відповідає за виведення невеликого фрагмента HTML-коду, який можна використовувати повторно.

Рисунок 2.1 — Приклад вигляду функціонального компоненту

Компоненти є серцем усіх програм побудованих на React. Ці компоненти можуть бути вкладені в інші компоненти, щоб дозволити будувати складні комплексні програми з простих блоків. ReactJS використовує віртуальний механізм на основі DOM для заповнення даних у HTML DOM. Віртуальний DOM працює швидко, оскільки змінює лише окремі елементи DOM замість того, щоб щоразу перезавантажувати повний DOM.

Redux — бібліотека призначена для управління станами у програмах JavaScript. Всі стани компонентів та інші дані можуть зберегатися в одному місті — store. Потреба у бібліотеці виникла через ускладнену маштабованість програми при використанні стандартної системи роботи з станами у ReactJS.

React-redux — бібліотека що слугує для полегшення роботи з бібліотекою Redux у поєднанні із ReactJS.

Redux Saga — це бібліотека, яка покликана спростити та покращити побічні ефекти (тобто такі дії, як асинхронні операції, наприклад, завантаження даних, та так званні "брудні" дії, такі як доступ до браузерного кешу), зробити легкими в тестуванні і краще справлятися із помилками.

Для більш зручної роботи з бібліотекою Redux у браузері Google Chrome існує розширення Redux DevTools яке допомагає відстежувати дії та дивитися вміст сховища станів у той чи іншій момент часу.

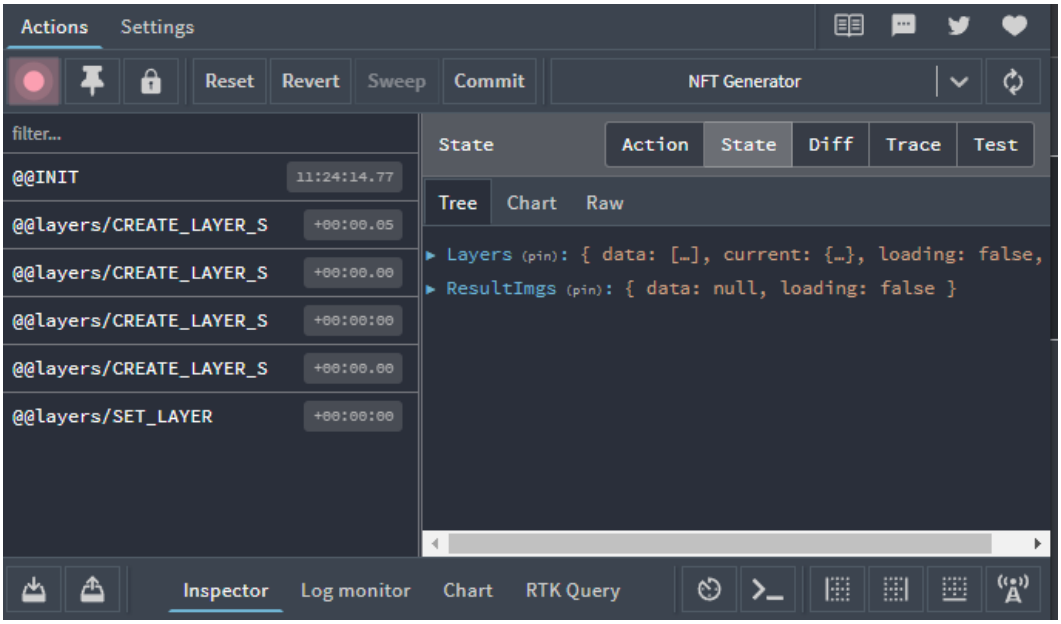

Рисунок 2.2 — Зовнішній вигляд Redux DevTools

Бібліотеки повністю підтримується мовою програмування TypeScript.

2.4 Візуальна середа розробки

Visual Studio Code — це редактор вихідного коду, створений Microsoft для Windows, Linux і macOS. Функції включають підтримку налагодження, виділення синтаксису, інтелектуальне завершення коду, фрагменти, рефакторинг коду та вбудований Git. Користувачі можуть змінювати тему, комбінації клавіш, параметри та встановлювати розширення, які додають додаткові функції [10].

У опитуванні розробників Stack Overflow 2021 Visual Studio Code був визнаний найпопулярнішим інструментом середовища розробника: 70% з 82 000 респондентів повідомили, що вони його використовують.

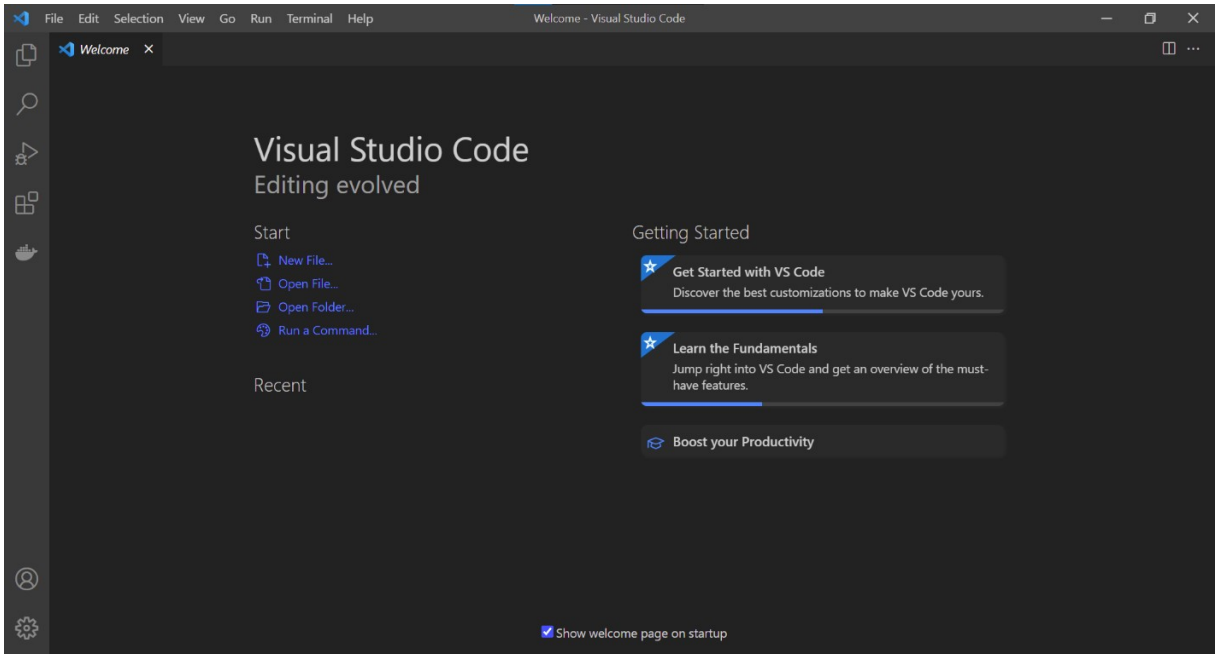

Рисунок 2.3 — Інтерфейс програми Visual Studio Code

2.5 Node Package Manager та встановлені пакети

Node Package Manager (npm) — це стандартний менеджер пакетів JavaScript. Він використовується для завантаження пакетів з хмарного сервера npm або для завантаження пакетів на ці сервери. Пакетами звуться інструменти чи бібліотеки на мові JavaScript або похідних від нього мов [11].

За допомогою npm для встановлених задач було необхідно завантажити ряд пакетів, вкажемо основні з них, а саме:

- Бібліотека React.js — див. розділ 2.3;
- Redux — див. розділ 2.3;
- React-redux — див. розділ 2.3;
- Redux-saga — див. розділ 2.3;
- Бібліотека styled-components — бібліотека для стилізації програм React. Вона дозволяє створювати власні компоненти за допомогою

написання самого CSS в JavaScript, приклад вигляду коду стилізованого компоненту можно побачити на рисунку 2.4;

- Бібліотека JSZip — бібліотека для формування архівів у JavaScript;
- Бібліотека file-saver — бібліотека для можливості збереження вихідних файлів на пристрій користувача у JavaScript;
- Бібліотека merge-images — бібліотека для роботи з файлами зображеннь у JavaScript;
- Бібліотека Canvas — бібліотека для роботи з 2D зображеннями у JavaScript;
- Бібліотека random-js — бібліотека для генерації математично правильніх випадкових чисел.

```
const WrapperS = styled.aside`
  padding: 10px;
  z-index: 1;
  position: fixed;
  left: 0;
  width: ${LEFT_SIDE_BAR_WIDTH};
  bottom: 0;
  height: calc(100vh - ${headerHeight});
  max-height: calc(100vh - ${headerHeight});
`;
```

Рисунок 2.4 — Приклад коду стилізованого компоненту

Слід зазначити що усі бібліотки повністю сумісні з TypeScript це виражається у наявності в цих бібліотеках типізованих версій функціоналу.

2.6 Причини вибору технологій реалізації

Мова програмування TypeScript була обрана через розширений функціонал та підвищена зручність розробки у порівнянні з JavaScript та наявний досвід розробки на мові програмування TypeScript. Велика кількість

сучасних комерційних вебдодатків віддають перевагу мові TypeScript, а не мові JavaScript через можливісь більш ефективного, а тому і більш швидкого і дешевого, поширення функціоналу та подальшої підтримки великого вебдодатку завдяки суровій типізації.

Бібліотека ReactJS була обрана через можливість ефективно реалізувати інтерфейс та функціонал вебдодатку через функціональні компоненти та особисту заінтересованість у здобутті додаткового досвіду роботи з бібліотекою ReactJS.

Редактор коду Visual Studio Code було обрано задля швидкої роботи, код не потребує повноцінної середи розробки адже буде виконуватися безпосередньо у браузері.

Усі встановлені бібліотеки були обрані на основі особистого досвіду, необхідного функціоналу та за відкритою статистикою найбільшої довіри та популярності серед користувачів npm.

2.7 Висновки за розділом

Розглянувши засоби створення вебдодатків, було прийнято рішення створювати вебдодаток за допомогою мови програмування TypeScript, з використанням бібліотеки ReactJS.

РОЗДІЛ 3
ПРОГРАМНЕ РІШЕННЯ

3.1 Початок розробки вебдодатку

Кожен вебдодаток розроблений із використанням React має в собі ключовий програмний складовий файл під назвою «package.json». Саме тут знаходяться ключові параметри розробленого проєкту, код даного файлу зображено на лістингу коду 3.1.

Лістинг коду 3.1 — Файл package.json

```
{
  "name": "nft_generator",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "animate.css": "^4.1.1",
    "canvas": "^2.9.1",
    "file-saver": "^2.0.5",
    "jszip": "^3.10.0",
    "merge-images": "^2.0.0",
    "random-js": "^2.1.0",
    "rc-slider": "^10.0.0",
    "react": "^17.0.1",
    "react-dom": "^17.0.1",
    "react-dropzone": "^14.2.1",
    "react-papaparse": "^4.0.2",
    "react-redux": "^7.2.2",
    "react-router-dom": "^5.2.0",
    "react-scripts": "4.0.3",
```

```
  "redux": "^4.0.5",
  "redux-devtools-extension": "^2.13.8",
  "redux-saga": "^1.1.3",
  "styled-components": "^5.2.1",
  "typesafe-actions": "^5.1.0",
  "typescript": "4.4.4",
  "uuid": "^8.3.2",
  "web-vitals": "^1.1.0"
},
"scripts": {
  "start": "PORT=5001 react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test",
  "eject": "react-scripts eject"
},
"eslintConfig": {
  "extends": [
    "react-app",
    "react-app/jest"
  ]
},
"browserslist": {
  "production": [
    ">0.2%",
    "not dead",
    "not op_mini all"
  ],
  "development": [
    "last 1 chrome version",
    "last 1 firefox version",
```

```
      "last 1 safari version"
    ]
  },
  "devDependencies": {
    "@testing-library/jest-dom": "^5.11.9",
    "@testing-library/react": "^11.2.5",
    "@testing-library/user-event": "^12.8.0",
    "@types/file-saver": "^2.0.5",
    "@types/jest": "^26.0.20",
    "@types/merge-images": "^1.2.1",
    "@types/node": "^14.14.31",
    "@types/react": "^18.0.9",
    "@types/react-dom": "^17.0.1",
    "@types/react-redux": "^7.1.24",
    "@types/react-router-dom": "^5.1.7",
    "@types/styled-components": "^5.1.7",
    "react-error-overlay": "6.0.9"
  }
}
```

Основний об’єкт файлу містить ключові поля, а саме:

- «name» - назва проєкту;
- «version» - версія проєкту;
- «private» - тип репозиторію;
- «dependencies» - основні бібліотеки проєкту;
- «scripts» - команди для консолі npm;
- «eslintConfig» - конфігурація інструменту для аналіза якості коду;
- «browserslist» - список підримуємих браузерів;
- «devDependencies» - бібліотеки для розробника, у нашому випадку більшість з них це типізовані версії встановлених бібліотек для зручної роботи із ними у TypeScript.

Щоб почати розробку заданої системи за тематикою дипломної роботи, необхідно встановити залежності через команду npm i та викликати створену у «scripts» команду npm start яка запустить React додаток за вказаним локальним портом, а саме за адресою «http://localhost:5001/».

3.2 Структура та код вебдодатку

Структура розробленого проекту зображена на рисунку 3.1.

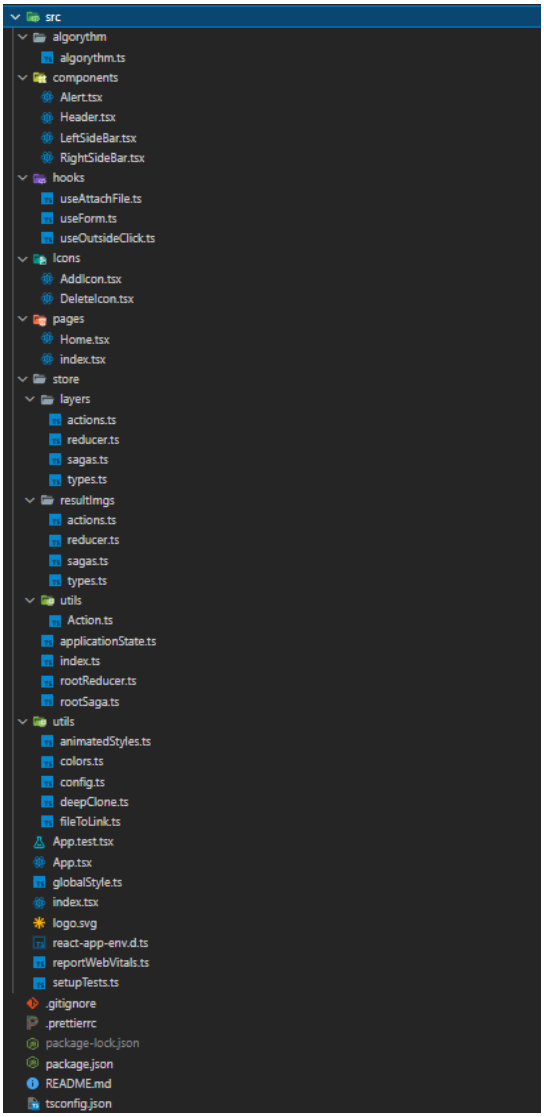

Рисунок 3.1 — Вид структури коду розробленного додатку

Основою будь якого додатку є файли «index.tsx» та «App.tsx» у папці «src». У файлі «index.tsx» підєднано store що знаходиться у папці «store» та глобальні стилі вебдодатку «globalStyles.ts»,

Лістинг коду 3.2 — Файл index.tsx

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Provider } from 'react-redux';
import { BrowserRouter as Router } from 'react-router-dom';
import { GlobalStyle } from './globalStyle';
import App from './App';
import store from './store';

ReactDOM.render(
<React.StrictMode>
 <Provider store={store}>
<Router>
<GlobalStyle />
<App />
</Router>
</Provider>
</React.StrictMode>,
document.getElementById('nftGenerator')
);
```

Лістинг коду 3.3 — Файл App.tsx

```
import { Switch } from 'react-router-dom';
import { LeftSideBar } from './components/LeftSideBar';
```

```
import { RightSideBar } from './components/RightSideBar';
import * as Screens from './pages';

function App() {
  return (
    <>
      <LeftSideBar />
      <RightSideBar />
      <Switch>
        <Screens.Home />
      </Switch>
    </>
  );
}

export default App;
```

Основний функіонал вебдодатку реалізовано за допомогою функціональних компонентів React. Для зручності та пришвидшення розробки зовнішнього вигляду цих компонентів були створені наступні файли:

- «config.ts» - файл з встановленими у змінні стильові значення необхідні для їх прямого застосування у компонентах та динамічного розрахунку розмірів сусідніх компонентів, код даного файлу зображено на лістингу 3.4;
- «colors.ts» - файл з основними кольорами додатку, що представлені у вигляді змнінниїх та функцій, код даного файлу зображено на лістингу 3.5;
- «globalStyles.tsx» - файл що містить основні стилізовані компоненти додатку та глобальний стилізований компонент у якому визначені основні стилі сторінки вебдодатку.

Лістинг коду 3.4 — Файл config.ts

```
export const FONT_FAMILY = `'Lato', sans-serif`;

export const DEFAULT_PADDING = '24px';

export const LEFT_SIDE_BAR_WIDTH = '350px';

export const RIGHT_SIDE_BAR_WIDTH = '350px';

export const HEADER_HEIGHT = '0px';
```

Лістинг коду 3.5 — Файл colors.ts

```
export const COLORS = {
  white: '#fffafa',
  default: '#36393f',
  secondary: '#b9bbbe',
  accent: '#36393f',
  dark: '#202225',
  accentHover: '#dcddde',
  danger: '#ff5e5e',
  dangerActive: '#b90404',
};
export const COLORS_RGBA = {
  secondary: (opacity: number = 1) => `rgba(185, 187, 190, ${opacity})`,
  default: (opacity: number = 1) => `rgba(54, 57, 63, ${opacity})`,
  accent: (opacity: number = 1) => `rgba(54, 57, 63, ${opacity})`,
  danger: (opacity: number = 1) => `rgba(255, 94, 94, ${opacity})`,
  dark: (opacity: number = 1) => `rgba(32, 34, 37, ${opacity})`,
  white: (opacity: number = 1) => `rgba(255, 250, 250, ${opacity})`,
};
```

Основною сторінкою став функціональній компонент Home, а поверх нього будуть розміщені компоненти LeftSideBar та RightSideBar, виклик яких можна побачити у файлі «App.tsx» (Лістинг коду 3.3 — Файл App.tsx).

Почнемо з компоненту LeftSideBar, у ньому реалізовано функціонал переліку існуючих, додання нових шарів зображень та переключення поточного шару натисканням потрібного у списку, саме до поточногу шару йде завантаження зображень з компоненту Home.

Відображення списку існуючих шарів відбувається у вигляді переліку компонентів у яких вказана назва шару, кількість завантажених до шару

зображень та шанс появи шару у вихідному зображенні у відсотках який користувач маж можливість налаштувати у компоненті RightSideBar. На прикінці списку є елемент який містить у собі поле введення ім’я шару та кнопку створення, створення новго шару можливе за умови не пустго поля імені шару, усі інші необхідні параметрі заповнюються автоматично.

Шари зображень створюються за моделлю даних, яку наведено у лістингу коду 3.6. За замовчанням буде створенно 4 таких шари після завантаження компоненту Home і надана можливість додавати та видаляти їх. Масив шарів за замовчанням відображено на лістингу коду 3.7.

Додавання та видалення відбувається функцією useDispatch — стандартної функції взаємодії зі store, яка забезпечить занесення створенного шару до нього, для створення використаємо action-об’єкт CreateLayer, а для видалення action-об’єкт DeleteLayer, їх наведено у лістингу коду 3.10.

Вигляд компоненту LeftSideBar наведено на рисунку 3.2.

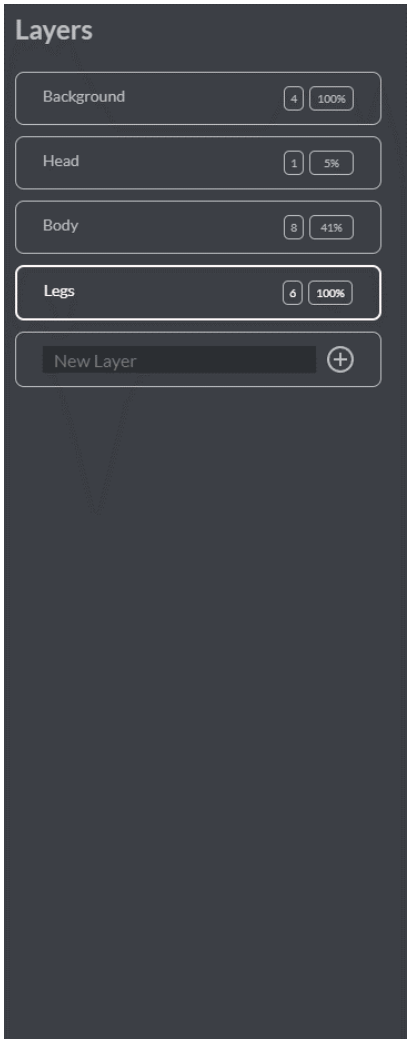

Рисунок 3.2 — Зовнішній вигляд компоненту LeftSideBar

Лістинг коду 3.6 — Модель даних TLayer

```
export interface TLayer {
  name: string;
  id: string;
  data: TAsset[];
  chance?: number;
}
```

Лістинг коду 3.7 — Константа з шарами за замовчанням

```
const defaultLayers: TLayer[] = [
  { name: 'Background', id: uuidv4(), chance: 1, data: null },
  { name: 'Head', id: uuidv4(), chance: 1, data: null },
  { name: 'Body', id: uuidv4(), chance: 1, data: null },
  { name: 'Legs', id: uuidv4(), chance: 1, data: null },
];
```

Задля створення шару формується відповідний об’єкт, що містить наступні поля:

- name – поле з іменем шару, для нього береться ім’я введене користувачем у полі введення при створенні шару, потріблне для подальшого відображення у додатку;
- id – поле з унікальним id сгенерованим за допомогою бібліотеки «uuid», потрібне для подальшої роботи із шаром та використанні його в якості унікального ключа при відображенні зображення у інтерфейсі;
- data – поле що безпосередньо містить у собі масив об’єктів що репрезентують завантажені користувачем зображення за моделлю даних, яку наведенно у лістингу коду 3.12;
- chance – поле що містить цифру, потрібне для подальшої репрезентації шансу використання шару у алгоритмі генерації.

Код компоненту LeftSideBar наведено у лістингу 3.8.

Лістинг коду 3.8 — Код функціонального компоненту LeftSideBar

```
export const LeftSideBar: React.FC<LeftSideBarProps> = () => {
const { Layers } = useSelector((store: AppStore) => store);
const [addValue, setAddValue] = React.useState('');
const dispatch = useDispatch();
```

```
const addHandler = () => {
    if (addValue) {
      dispatch(
        CreateLayer({
          name: addValue,
          id: uuidv4(),
          data: null,
          chance: 1,
        })
      );
      setAddValue('');
    }
  };
  const layerClickHandler = (layer: TLayer) => {
    if (Layers.current.id !== layer.id) dispatch(SetCurrentLayer(layer));
  };
  return (
    <WrapperS>
      <h1>Layers</h1>
      <ListWrapperS>
        {Layers.data &&
          Layers.data.map((layer) => {
            return (
              <ListItemS
                isSelected={Layers.current.id === layer.id}
                onClick={() => layerClickHandler(layer)}
                key={layer.id}
              >
                {layer.name}
```

```
            <NumbersWrapperS>
              <CountWrapperS>
                {layer.data ? layer.data.length : 0}
              </CountWrapperS>
              <PercentWrapperS>
                {layer.chance ? `${layer.chance * 100}%` : 'Error'}
              </PercentWrapperS>
            </NumbersWrapperS>
          </ListItemS>
        );
      })}
      <ListItemS isSelected={false}>
        <AddNewInputS
          placeholder="New Layer"
          onChange={(e) => {
            setAddValue(e.currentTarget.value);
          }}
          value={addValue}
        ></AddNewInputS>
        <AddIconWrapperS onClick={addHandler}>
          <AddIcon />
        </AddIconWrapperS>
      </ListItemS>
    </ListWrapperS>
  </WrapperS>
 );
};
```

Далі розглянемо компонент Home, він містить у собі можливість завантаження вихідних зображень користувача з привязкою до поточного шару

зображень для їх відображення та подальшої роботи з ними, а також змінну що містить у собі певну кількість пустих шарів за замовчанням. Для завантаження файлів необхідно перетягнути їх у відповідне поле або натиснути на поле та обрти необхідні файли через файлову систему. Приймаються лише файли відповідних форматів: «.png», «.jpeg», «.jpg», «.webp». На випадок якщо користувач буде намагатися завантажити файли з недопустимими розширеннями він отримає помилку із відповідним поясненням.

Повний код компоненту Home наведено у лістингу коду 3.9, його зовнішній вигляд та зовнішній вигляд із повідомленням про помилку відобраджено на рисунках 3.3 та 3.4 відповідно.

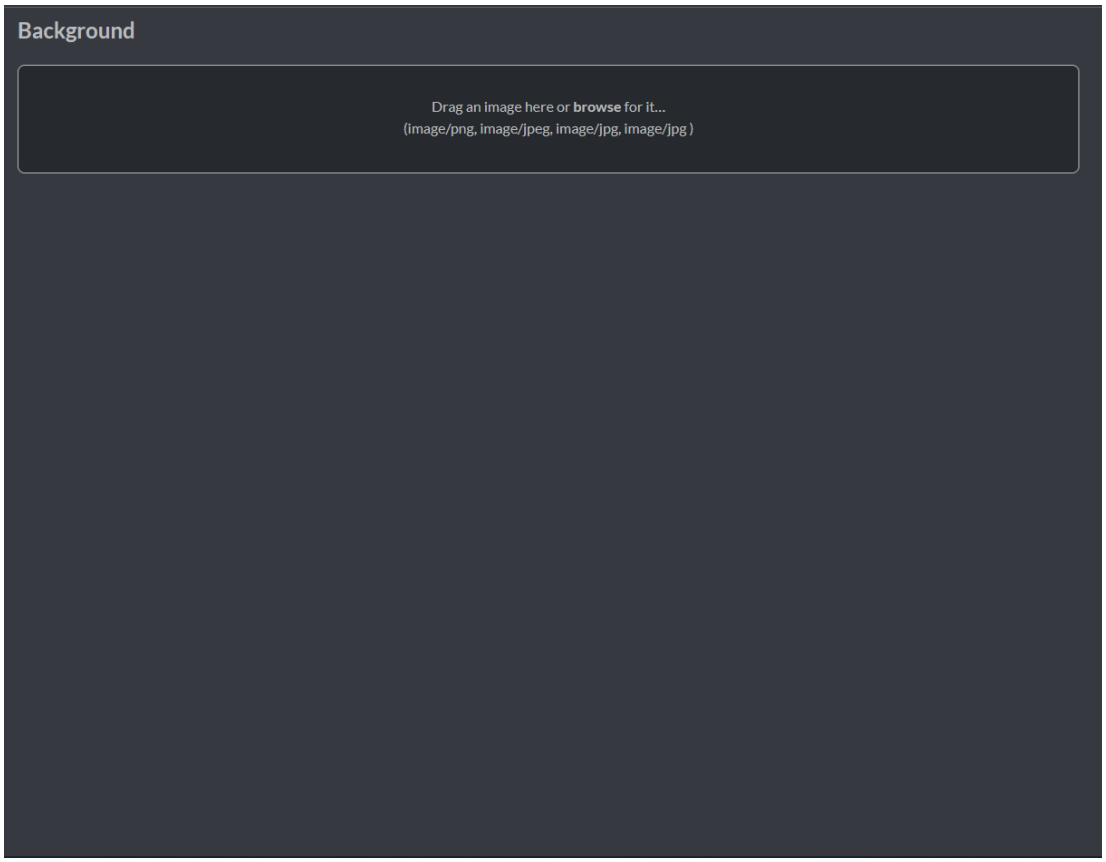

Рисунок 3.3 — Зовнішній вигляд компоненту Home

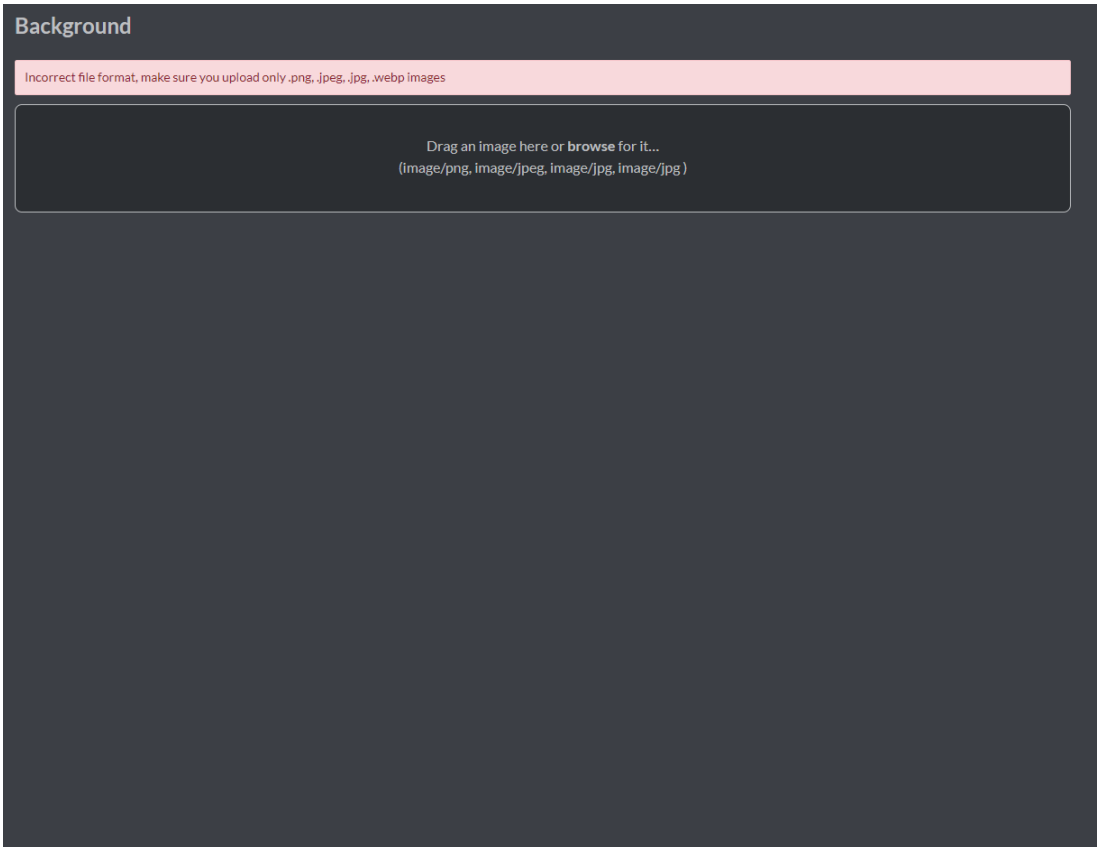

Рисунок 3.5 — Зовнішній вигляд повідомлення у компоненті Home

Лістинг коду 3.9 — Код функціонального компоненту Home

```
export const Home: React.FC<HomeProps> = () => {
  const { Layers } = useSelector((store: AppStore) => store);
  const dispatch = useDispatch();
  const hookAttach = useAttachFile();

  React.useEffect(() => {
    if (!Layers.data) {
      defaultLayers.forEach((layer) => {
        dispatch(CreateLayer(layer));
      });
      dispatch(SetCurrentLayer(defaultLayers[0]));
    }
    //eslint-disable-next-line
```

```
  }, [Layers]);
  const deleteAssetHandler = (id: string) => {
    const newLayers = Layers.data;
    const newAssets: TAsset[] = Layers.current.data;
    const assetIdx = newAssets.findIndex((asset) => asset.id === id);
    newAssets.splice(assetIdx, 1);
    newLayers.find((layer) => layer.id === Layers.current.id).data = newAssets;
    dispatch(EditLayer(newLayers));
    dispatch(
                SetCurrentLayer(newLayers.find((layer) => layer.id ===
Layers.current.id))
    );
  };
  return (
    <MainWrapperS>
      <h1>{Layers.current && Layers.current.name}</h1>
      <div>
        <Alert text={hookAttach.errorText} />
      </div>
      <ImgWrapperS>
        {Layers.current &&
          Layers.current.data &&
          Layers.current.data.map((asset) => {
            return (
              <ImgItmS key={asset.id}>
                <img src={asset.data} alt={asset.name}></img>
                <ImageTextS>{asset.name}</ImageTextS>
                <DeleteWrapperS onClick={() => deleteAssetHandler(asset.id)}>
                  <DeleteIcon />
                </DeleteWrapperS>
```

```
            </ImgItmS>
          );
        })}
      </ImgWrapperS>
      <DropWrapperS
        {...hookAttach.getRootProps()}
        disabled={hookAttach.loading}
      >
        <input {...hookAttach.getInputProps()} />
        {hookAttach.isDragActive ? (
          <p>Drop the img here ...</p>
        ) : (
          <p>
            Drag an image here or <span>browse</span> for it…
          </p>
        )}
      </DropWrapperS>
    </MainWrapperS>
  );
};
```

Дані завантажені користувачем зберігаються у store, основними складовими якого є файли rootReducer.ts, rootSaga.ts, applicationState.ts. Ці файли об’єднуються у файлі index.ts та створюють той самий єдиний глобальний store, більш детально зупинимося на кожному із них та на складовії що він об’єднує. У кожного елемнту store є зазвичай від двох до чотирьох файлів, а саме:

- «actions.ts» - файл з функціями які повертають action – об’єкт для виклику певної взаємодії з конкретним редьюсером, приклад змісту файлу наведено у лістингу коду 3.10;
- «reducer.ts» - файл з reducer – функцією який та стандартними значеннями для вмісту певного елементу store, reducer – функція являє собою просту функцію яка в залежності від типу action – об’єкту створює новий store, приклад змісту такого файлу наведено

у лістингу коду 3.11. Такі reducer – функції у собі обєднує файл rootReducer.ts, файл наведено у лістингу коду 3.12;

- «types.ts» - файл що зазвичай містить певні інтерфейси об’єктів, модель даних елемнту store та об’єкт з переліком типів action – об’єктів, його наявність є опціональною але є популярною практикою, приклад змісту такого файлу наведено у лістингу коду 3.13. Зазвичай моделі елементів store об’єднуються в один інтерфейс який відображає собою поточний store, такий інтерфейс прийнято окремо розміщувати у файлі applicationState.ts, файл наведено у лістингу коду 3.14;
- «sagas.ts» - файл що містить функції які виконуються після виклику того чи іншого action-об’єкту, зазвичай використовуються при потребі роботи із WEB-API, його наявність є опціональною і у проекті такі файли відсуті. Такі файли зазвичай об’єднують у єдину функцію у файлі rootSaga.ts.

Лістинг коду 3.10 — Файл actions.ts

```
import { action } from 'typesafe-actions';
import ActionTypes, { TLayer } from './types';

export const CreateLayer = (payload: TLayer) =>
  action(ActionTypes.CREATE_LAYER_S, payload);

export const DeleteLayer = (payload: TLayer[]) =>
  action(ActionTypes.DELETE_LAYER_S, payload);

export const EditLayer = (payload: TLayer[]) =>
  action(ActionTypes.EDIT_LAYER_S, payload);

export const SetMaxAmount = (payload: number) =>
  action(ActionTypes.SET_MAX_AMOUNT, payload);

export const SetCurrentLayer = (payload: TLayer) =>
  action(ActionTypes.SET_LAYER, payload);

export const cleanLayers = () => action(ActionTypes.CLEAN_UP, null);
```

Лістинг коду 3.11 — Файл reducer.ts

```
import { Reducer } from 'redux';
import LayerActionTypes, { TLayersState } from './types';
export const initialState: TLayersState = {
  data: null,
  current: null,
  loading: false,
  maxAmount: 0,
  errors: undefined,
```

```
};
const reducer: Reducer<TLayersState> = (state = initialState, action) => {
  switch (action.type) {
    case LayerActionTypes.CREATE_LAYER_S:
      return {
        ...state,
        data: state.data ? [...state.data, action.payload] : [action.payload],
      };
    case LayerActionTypes.CREATE_LAYER_E:
      return { ...state, errors: action.payload };
    case LayerActionTypes.DELETE_LAYER_S:
      return { ...state, data: action.payload };
    case LayerActionTypes.DELETE_LAYER_E:
      return { ...state, errors: action.payload };
    case LayerActionTypes.EDIT_LAYER_S:
      return { ...state, data: action.payload };
    case LayerActionTypes.EDIT_LAYER_E:
      return { ...state, errors: action.payload };
    case LayerActionTypes.SET_LAYER:
      return {
        ...state,
        current: action.payload,
      };
    case LayerActionTypes.SET_MAX_AMOUNT:
      return {
        ...state,
        maxAmount: action.payload,
      };
    default:
      return state;
  }
};

export { reducer as LayerReducer };
```

Лістинг коду 3.12 — Файл rootReducer.ts

```
import { combineReducers } from 'redux';
import { LayerReducer } from './layers/reducer';
const rootReducer = combineReducers({
  Layers: LayerReducer,});
```

```
export default rootReducer;
```

Лістинг коду 3.13 — Файл types.ts

```
export interface TLayer {
  name: string;
  id: string;
  data: TAsset[];
  chance: number;
}
export interface TAsset {
  name: string;
  id: string;
  data: string;
  weight: number;
}
export interface TLayersState {
  readonly loading: boolean;
  data: TLayer[] | null;
  current: TLayer | null;
  maxAmount: number;
  readonly errors?: string | undefined;
}

enum LayerActionTypes {
  CREATE_LAYER_S = '@@layers/CREATE_LAYER_S',
  CREATE_LAYER_E = '@@layers/CREATE_LAYER_E',

  EDIT_LAYER_S = '@@layers/EDIT_LAYER_S',
  EDIT_LAYER_E = '@@layers/EDIT_LAYER_E',
```

```
  DELETE_LAYER_S = '@@layers/DELETE_LAYER_S',
  DELETE_LAYER_E = '@@layers/DELETE_LAYER_E',

  SET_LAYER = '@@layers/SET_LAYER',

  SET_MAX_AMOUNT = '@@layers/SET_MAX_AMOUNT',

  CLEAN_UP = '@@layers/CLEAN_UP',
}

export default LayerActionTypes;
```

Лістинг коду 3.14 — Файл applicationState.ts

```
import { TLayersState } from './layers/types';

export interface AppStore {
  Layers: TLayersState;
}
```

Для завантаження файлів за методом drag-n-drop на сторінці Home, їх подальшої обробки та занесенню у store було створено функцію useAttachFile, код якої наведено у лістингу коду 3.17. Функція створена із застосуванням бібліотеки «react-dropzone» яка забезпечить завантаження файлів потрібних типів, та використанням функції useDispatch — стандартної функції взаємодії зі store, яка забезпечить занесення створенних із файлів об’єктів до нього, для використаємо action-об’єкти SetMaxAmount для занесення максимально можливої кількості унікальних вихідних зображень до store, EditLayer та SetCurrentLayer для додання зображень користувача до поточного шару зображень, ці функції наведено у лістингу коду 3.10.

Максимальна кількість унікальних вихідних зображень необхідна для алгоритму генерації, вона вираховується перемноженням кількості

завантажених користувачем зображень з кожного шару та необхідна для встановлення максимальної межі при введенніі потрібної користувачеві кількості вихідних зображень у відповідному полі.

Зображення користувача зберігаються у store за моделлю даних яку наведено у лістингу коду 3.15. Після завантаження зображень до поточного шару користувачем для кожного зображення формується відповідний об’єкт що містить наступні поля:

- name – поле з іменем зображення, для нього береться ім’я завантажуємого користувачем файлу, потріблне для подальшого відображення у додатку;
- id – поле з унікальним id сгенерованим за допомогою бібліотеки «uuid», потрібне для подальшої роботи із зображенням та використанні його в якості унікального ключа при відображенні зображення у інтерфейсі;
- data – поле що безпосередньо містить у собі зображення. Через те що зберігання зображення в форматі base64 у store не є ефективним було прийнято рішення про зберігання зображення у форматі посилання на нього, перетворення у такий формат відбувається за допомогою функції fileToLink яку наведено у лістингу коду 3.16.
- weight – поле що містить так звану «вагу» для подальшого визначення частоти вибору цього зображення у алгоритмі генерації.

Лістинг коду 3.15 — Модель данних TAsset

```
export interface TAsset {
  name: string;
  id: string;
  data: string;
  weight: number;
```

```
}
```

Лістинг коду 3.16 — Функція fileToLink

```
export const fileToLink = (file: File) => {
  const localImageUrl = window.URL.createObjectURL(file);
  return localImageUrl;
};
```

Лістинг коду 3.17 — Код функції useAttachFile

```
export const useAttachFile = () => {
  const { Layers } = useSelector((store: AppStore) => store);
  const [errorText, setErrorText] = React.useState('');
  const [isFileLoaded, setIsFileLoaded] = React.useState(false);
  const [loading, setLoading] = React.useState(false);
  const dispatch = useDispatch();
  const hookParse = usePapaParse();
  const onDrop = React.useCallback(
    (acceptedFiles) => {
      setErrorText('');
      if (!acceptedFiles[0])
        return setErrorText(
          'Incorrect file format, make sure you upload only .png, .jpeg, .jpg, .webp
images'
        );
      const newLayers = Layers.data;
      let maxAmount = null;
      const newAssets: TAsset[] =
        Layers.current && Layers.current.data ? Layers.current.data : [];
      acceptedFiles.forEach((file) => {
```

```
if (!file) return setErrorText('File is undefined');
if (
  !file.name.includes('.png') &&
  !file.name.includes('.jpg') &&
  !file.name.includes('.jpeg') &&
  !file.name.includes('.webp')
)
  return setErrorText(
                    'Incorrect file format, make sure you upload only .png, .jpeg, .jpg, .webp images'
  );
const fileReader = new FileReader();
var image = new Image();
const handleFileReader = () => {
  const content = fileReader.result;
  const results = hookParse.readString(content as string, null) as any;
  if (results.data) setIsFileLoaded(true);
};
fileReader.onloadend = handleFileReader;
fileReader.readAsText(file);
image.src = fileToLink(file);
newAssets.push({
  name: file.name,
  id: uuidv4(),
  data: fileToLink(file),
  weight: 1,
});
});
newLayers.find((layer) => layer.id === Layers.current.id).data =
  newAssets;
```

```
      newLayers.forEach((layer) => {
        if (layer.data && layer.data[0])
          maxAmount = maxAmount
            ? maxAmount * layer.data.length
            : layer.data.length;
        console.log(maxAmount);
      });
      dispatch(SetMaxAmount(maxAmount));
      dispatch(EditLayer(newLayers));
    },
    [hookParse, Layers, dispatch]
  );
  const { getRootProps, getInputProps, isDragActive } = useDropzone({
    onDrop,
    accept: {
      'image/png': ['.png', '.jpeg', '.jpg', '.webp'],
    },
  });
  return {
    getRootProps,
    getInputProps,
    isDragActive,
    errorText,
    isFileLoaded,
    loading,
    setErrorText,
    setLoading,
  };
};
```

Після завантаження зображень користувачем, їх відповідної обробки та розміщення у store, компонент Home та компонент LeftSideBar отримають цю інформацію та оновляться. Оновлений вигляд компонентів можна побачити на малюнках (Рисунок 3.4 – 3.5).

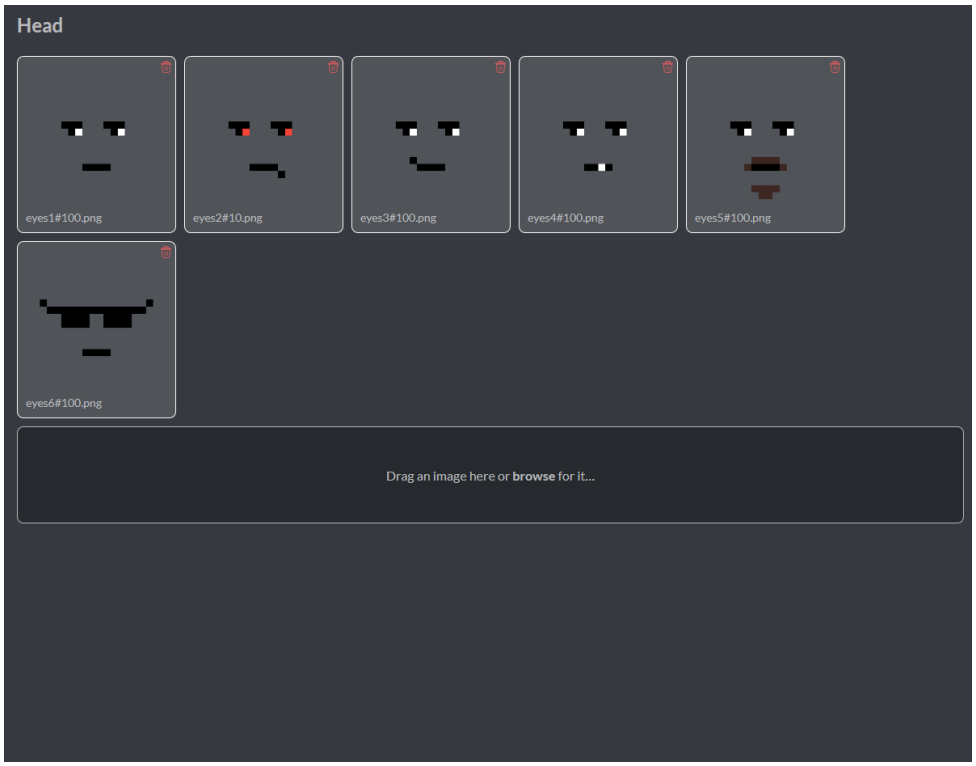

Рисунок 3.5 – Вигляд оновленого компоненту Home

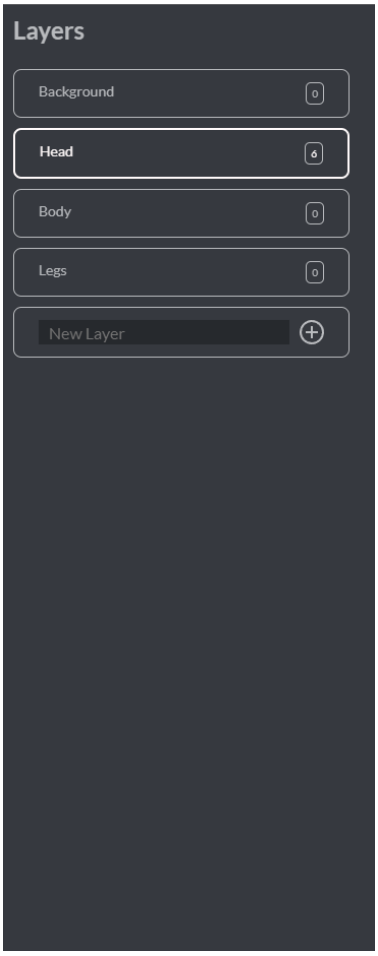

Рисунок 3.6 – Вигляд оновленого компоненту LeftSideBar

Останнім розглянемо компонент RightSideBar, у ньому будуть знаходитися налаштування параметрів назви та шансу поточного шару, кнопка «Delete Layer», поле введення потрібної користувачеві кількості вихідних зображнь та кнопка «Generate». Кінцевий зовнішній вигляд компоненту RightSideBar можна побачити на рисунку 3.6.

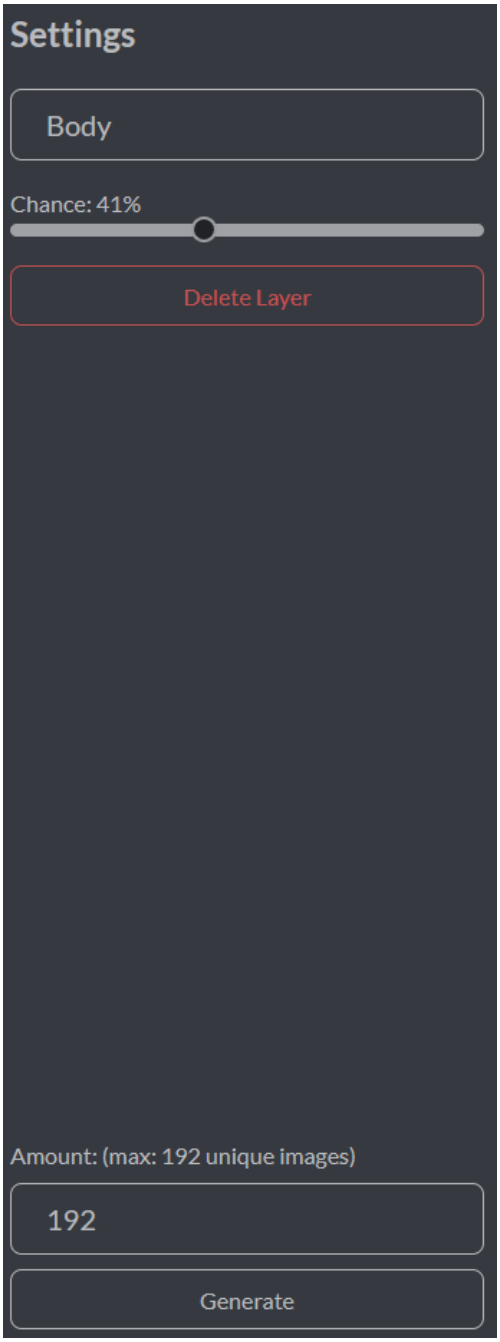

Рисунок 3.7 – Вигляд компоненту RightSideBar

Більш детально розглянемо складові цього компоненту.

Поле введення з ім’ям шару оновлює відповідні данні у store одразу після введення, ця дія оновлює компоненти LeftSideBar і Home у місцях де ім’я шару використовується.

Поле введення шансу реалізовано у вигляді слайдеру за допомогою бібліотеки «rc-slider» де мінімальне значення 1 відсоток, а максимальне 100 відсотків, але сам параметер шансу зберігається у спрощеному вигляді де

мінімальне значення це 0 а максимальне 1, при переміщенні повзунка відбувається конвертація з поточного значення параметру до спрощенного вигляду та його занесення до store після завершення переміщення.

Кнопка «Delete Layer» видаляє поточний шар разом з усіма завантаженими до нього зображеннями, видалення можливе лише за наявності ще мінімум одного іншого шару, це потрібно для запобігання візуальних помилок у вебдодатку, при спробі видалення останнього шару буде виведене повідомлення про помилку, яке можна побачити на рисунку 3.7.

Поле введення потрібної кількості вихідних зображень потрібне для алгоритму генерації та має обмеження у вигляді максимально можливої кількості унікальних вихідних зображень.

Кнопка «Generate» викликає функцію алгоритму передаючи туди усі необхідні параметри, у тому числі й потрібну користувачеві кількість вихідних зображень, ця кнопка є заблокованою, якщо користувач не завантажив жодного зображення або максимально можлива кількість вихідних хображень не перевищує одного, зовнішній вигляд заблокованої кнопки можна побачити на рисунку 3.7.

Повний код компоненту RightSideBar наведено у лістингу коду 3.18

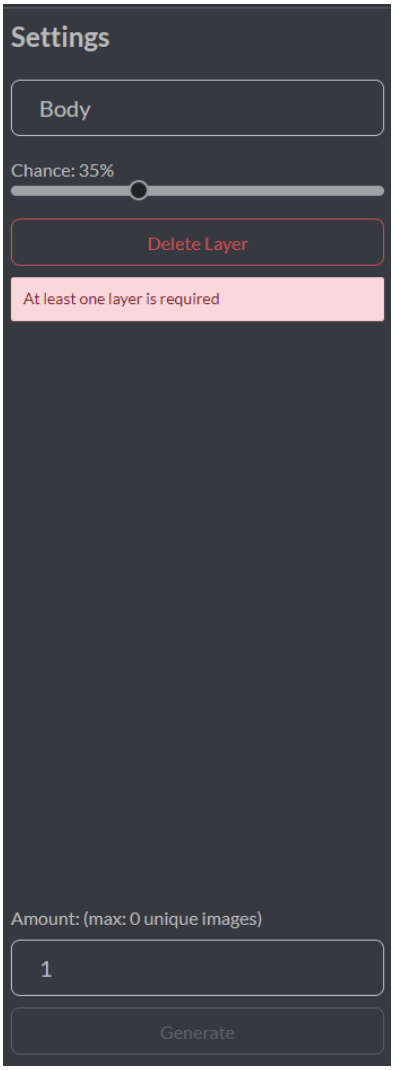

Рисунок 3.8 – Компонент RightSideBar з помилкою видалення та заблокованою кнопкою «Generate»

Лістинг коду 3.18 — Код функціонального компоненту RightSideBar

```
export const RightSideBar: React.FC<RightSideBarProps> = React.memo(() => {
  const { Layers } = useSelector((store: AppStore) => store);
  const [rangeValue, setRangeValue] = React.useState(0);
  const [nameValue, setNameValue] = React.useState('');
  const [currentImage, setCurrentImage] = React.useState(0);
  const [error, setError] = React.useState('');
  const [amountValue, setAmountValue] = React.useState(1);
```

```
const dispatch = useDispatch();

const deleteHandler = () => {
  if (!Layers.data) return setError('Unknown error');
  if (Layers.data.length <= 1)
    return setError('At least one layer is required');
  const newLayers = Layers.data;
  const currentIdx = newLayers.findIndex(
    (layer) => layer.id === Layers.current.id
  );
  newLayers.splice(currentIdx, 1);
  dispatch(DeleteLayer(newLayers));
  dispatch(SetCurrentLayer(newLayers[0]));
};
React.useEffect(() => {
  if (Layers.current) {
    setRangeValue(Layers.current.chance * 100);
    setNameValue(Layers.current.name);
  }
}, [Layers.current]);
return (
  <RightSideBarWrapperS>
    <SettingsWrapperS>
      <h1>Settings</h1>
      <NameInputS
        value={nameValue}
        onChange={(e) => {
          setNameValue(e.currentTarget.value);
          const newLayers = Layers.data;
          newLayers.find((layer) => layer.id === Layers.current.id).name =
```

```
      e.currentTarget.value;
    dispatch(EditLayer(newLayers));
  }}
></NameInputS>

<SliderWrapperS>
  <LabelS>Chance: {rangeValue}%</LabelS>
  <Slider
    range
    min={1}
    max={100}
    value={rangeValue}
    onChange={(e) => {
      setRangeValue(e as number);
    }}
    onAfterChange={(e) => {
      const newLayers = Layers.data;
      newLayers.find((layer) => layer.id === Layers.current.id).chance =
        (e as number) / 100;
      dispatch(EditLayer(newLayers));
    }}
    handleStyle={{
      height: 18,
      width: 18,
      borderRadius: '50%',
      backgroundColor: `${COLORS.dark}`,
      opacity: 1,
    }}
    ariaLabelForHandle={'123'}
  />
```

```
          </SliderWrapperS>
                                <DeleteButtonS    onClick={deleteHandler}>Delete
Layer</DeleteButtonS>
          {error && <Alert text={error} />}
        </SettingsWrapperS>
        <GeneraterWrapperS>
                        <LabelS>Amount:  (max:  {Layers.maxAmount}  unique
images)</LabelS>
          <AmountInputS
           value={amountValue}
           onChange={(e) => {
            if (
              e.target.value &&
              e.target.value !== '0' &&
              Layers.maxAmount >= parseInt(e.target.value)
            )
              setAmountValue(parseInt(e.target.value));
           }}
           type="number"
           step={50}
           min={1}
           max={Layers.maxAmount}
          ></AmountInputS>
          <GenButtonS
           onClick={() => {
            if (Layers.data) {
              generateNfts(amountValue, Layers.data, setCurrentImage);
            }
           }}
           disabled={Layers.maxAmount <= 1}
```

```
          >
            {currentImage ? `${currentImage}/${amountValue}` : 'Generate'}
          </GenButtonS>
        </GeneraterWrapperS>
      </RightSideBarWrapperS>
    );
  });
```

Розмістивши усі розглянуті компоненти у файлі App.tsx отримаємо кінцевий вигляд вебдодатку, його можна побачити на малюнку 3.8. За допомогою такого розташування компонентів та їх стильових особливостях отримуємо сторінку де центральний компонент Home розташовано незалежно від бокових, це допоможе у ситуаціях коли користувач завантажить велику кількість зображень до певного шару, залишивши компоненти LeftSideBar та RightSideBar з усім їх функціоналом доступними користувачу, при цьому дозволяючи ним працювати із завантаженими зображеннями, як це зображено на рисунку 3.8

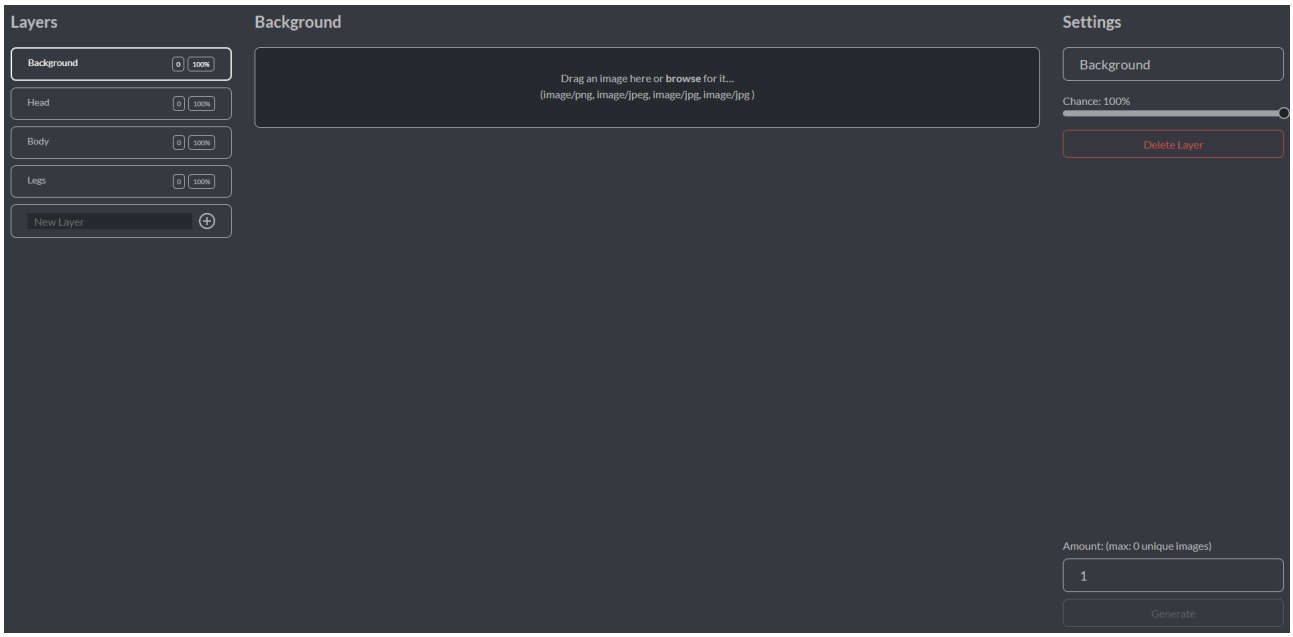

Рисунок 3.9 – Кінцевий вигляд вебдодатку

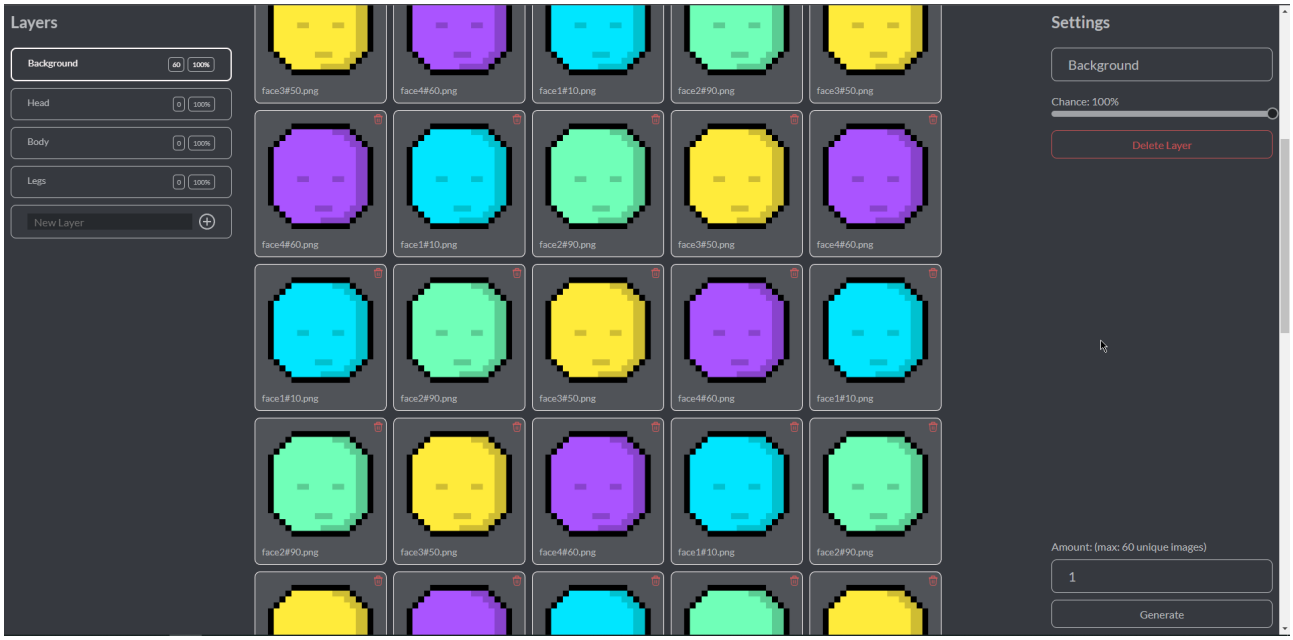

Рисунок 3.10 – Вигляд вебдодатку з великою кількістю завантажених зображень

3.3 Розробка алгоритму

Після того як уся візуально-функціональна частина додатку, розробка та імлементація моделей даних завершена можно безпосереднь перейти до розробки алгоритму.

Для генерації одного вихідного зображення необхідно зібрати випадкові зображення з усіх шарів та накласти зображення користувача один на одне по порядку у якому розташовані шари аби не перекрити одні елементи зображень іншими. Слід зазначити що через наявність у шарів такого параметру як «chance», який буде використано яе шанс потрапляння шару до вихідного зображення, потрібно перед тим як обрати шар обробити це значення відповідним чином.

Після створення таким чином одного вихідного зображення, алгоритм заносить його в змінну, а данні зображень користувача зя кого воно сгенеровано заносяться в JSON файл та переходить до генерації наступного зображення поки не досягне кількості необхідних вихідних зображень які передані до функції генерації у відповідному параметрі.

Генерація JSON файлу необхідна для ефективного порівняння вже згенерованих зображень із тим зображенням яке генеруються у поточний момент аби запобігти створенню дублікатів при генерації.

На останьому етапі генерації на основі усіх занесених до відповідної змінної зображень створюється архів, до якого окрім них додається файл з описом усіх комбінацій наявних у архіві вихідних зображень, такий файл необхідний деяким сайтам за NFT тематикою та може знадобитися користувачу. Такий архів зберігається користувачу на пристрій після чого алгоритм завершує свою роботу.

Спочатку створимо функцію для випадкового вибору шарів на основі параметру «chance» який відображає шанс вибору певного шару під час генерації. Функція повинна приймати масив існуючих шарів, далі за допомогою бібліотеки «random-js» створюємо змінну «mt» яка буде репрезентувати Вихор Марсена.

Вихор Мерсенна — генератор псевдовипадкових чисел (ГПВЧ), розроблений 1997 року японськими вченими Макото Мацумото і Такудзі Нісімурою. Вихор Мерсенна ґрунтується на властивостях простих чисел Мерсенна (звідси й назва) і забезпечує швидке генерування високоякісних за критерієм випадковості псевдовипадкових чисел.

Змінні «images» та «selectedTraits» будуть відображати усі обрані зображення користувача які будуть використані для подальшої генерації та данні цих зображень що будуть об’єднані та занесені у JSON файл на наступних кроках відповідно. Ці дві змінні і стануть результатами роботи фукції randomlySelectLayers.

Яке саме зображення буде обрані з конктретного шару буде визначено у функції pickWeighted результат роботи якої буде занесено у змінну selected та оброблено відповідним образом.

Повний код функції randomlySelectLayers відображено у лістингу коду 3.19

Лістинг коду 3.19 — Код функції randomlySelectLayers

```
const randomlySelectLayers = (layers: TLayer[]) => {
  const mt = MersenneTwister19937.autoSeed();

  let images = [];
  let selectedTraits = {};

  for (const layer of layers) {
    if (layer.data && bool(layer.chance)(mt)) {
      let selected = pickWeighted(mt, layer.data);
      selectedTraits[layer.name] = selected.name;
      images.push(selected.data);
    }
  }
  return {
    images,
    selectedTraits,
  };
};
```

Далі створимо функцію pickWeighted до якої передамо змінну «mt» що репрезентує Вихор Марсену та усі наявні у шарі зображення користувача. У кожного об’єкту що містить зображення користувача є параметр «weight», який відображає умовний шанс на вибір саме цього зображення. Результатом роботи цієї функції є об’єкт за моделлю даних TAsset яку наведено у лістингу коду 3.15, який містить обране зображення. Повний код функції pickWeighted наведено у лістингу коду 3.20.

Лістинг коду 3.20 — Код функції pickWeighted

```
function pickWeighted(mt, assets: TAsset[]) {
  const weightSum = assets.reduce((acc, asset) => {
    return acc + (asset.weight ?? 1.0);
  }, 0);

  const realNumber = real(0.0, weightSum, false)(mt);

  let summedWeight = 0.0;
  for (const asset of assets) {
    summedWeight += asset.weight ?? 1.0;
    if (realNumber <= summedWeight) {
      return asset;
    }
  }
}
```

Далі створимо функцію об’єднання обраних зображень у вихідне зображення та занесенню його у змінну архіву. Ця функція буде асинхронною та буде отримувати змінну формату «JSZip» який надано бібліотекою «jszip», масив обраних для генерації картинок користувача та поточний номер створюємого вихідного зображення.

За допомогою бібліотеки «merge-images» об’єднуємо масив зображень користувача в одне зображення шляхом накладання один на одне починаючи з початку массиву. Результат об’єднання буде представленно у форматі base64, від якого буде відділено необхідну частину, а саме – необхідні двійкові данні у ASCII-форматі, та на її основі сформуємо файл у змінній формату «JSZip» та

назвемо її за номером поточного вихідного зображення. Повний код цієї функції наведено у лістингу коду 3.21

Лістинг коду 3.21 — Код функції mergeLayersAndSave

```
async function mergeLayersAndSave(
  selectedImages,
  zip: JSZip,
  imgNumber: number
) {
  const image = await mergeImages(selectedImages, {
    Canvas: Canvas,
    Image: Image,
  });
  const idx = image.indexOf('base64,') + 'base64,'.length;
  const content = image.substring(idx);
  zip.file(`Nft-${imgNumber}.png`, content, {
    base64: true,
  });
}
```

Далі з використанням усіх вищезазначених функцій створимо основну функцію яку назвемо generateNfts, саме вона буде експортована і використана при натисканні на кнопку «Generate». У функцію буде передаватися необхідна кількість вихідних зображень, масив об’єктів шарів за моделлю даних «TLayers» яку наведено у лістингу коду 3.13 та функцію для оновлення стану у компоненті задля інформування користувача про прогресс генерації зображеннь.

У функції створимо сет у який будемо заносити дані про згенеровані зображення отримані з функції randomlySelectLayers та змінну zip формату

JSZip за допомогою якої буде реалізована архівація. За допомогою циклу досягаємо необхідної кількості повторення процессу генерації. Він включає у себе:

- занесення до стану номеру поточного генеруємого зображення
- отримання результату роботи функції «randomlySelectLayers» та занесення його до змінної
- створення змінної «traitsStr» що містить у собі інформацію про зображення обрані для генерації поточного вихідного зображення та її порівняння з існуючими. У випадку виявлення існуючого зображдення, яке було згенеровано за ідентичною комбінацією зображень йде повтор генерації зображення за поточним номеромб інакше «traitsStr» заноситься до сету існуючих а масив обраних зображенн користувача передаєься до функції «mergeLayersAndSave» для подальшого створення вихідного зображення та його збереження.
- Якщо поточне зображення останнє і генерацію завершено, до архіву zip додається файл «Traits.txt» що формується на основі сету інформації згенерованих зображень, після цього генерується архів формату «.zip» та завантажується користувачу на пристрій за допомогою функції «saveAs» бібліотеки «file-saver»

Повний код цієї функції наведено у лістингу коду 3.22.

Лістинг коду 3.22 — Код функції generateNfts

```
export const generateNfts = async (
  amount: number,
  layers: TLayer[],
  setCurrentImg: React.Dispatch<React.SetStateAction<number>>
) => {
```

```
  let generated = new Set();
  let zip = new JSZip();
  for (let tokenId = 1; tokenId <= amount; tokenId++) {
    setCurrentImg(tokenId);
    console.log(`Generating NFT #${tokenId} …`);
    let selection = randomlySelectLayers(layers);
    const traitsStr = JSON.stringify(selection.selectedTraits);

    if (generated.has(traitsStr)) {
      console.log('Duplicate detected. Retry …');
      tokenId--;
      continue;
    } else {
      generated.add(traitsStr);
      await mergeLayersAndSave(selection.images, zip, tokenId);
      if (tokenId === amount) {
        console.log('Finishing');
        zip.file('Traits.txt', JSON.stringify(Array.from(generated)));
        zip.generateAsync({ type: 'blob' }).then((content) => {
          saveAs(content, `GeneratedImages.zip`);
        });
        setCurrentImg(0);
      }
    }
  }
};
```

Пілся завершення розробки алгоритму генерації та створення необхідної функціїї додамо її виклик по натисканню на кнопку «Generate» у компоненті «RightSideBar» передаючи до неї усі необхідні данні.

3.4 Тестування функціоналу

Для тестування основного функціоналу вебдодатку були взяті 19 зображенньь з відкритого доступу.

Заповнимо та відредагуємо шари необхідним нам чином.

Перший шар – переіменований із «Background» у «Heads», містить чотири зображення і має стовідсотковий шанс появи у вихідному зображенні, це відображено на рисунку 3.10.

Рисунок 3.10 – Зображення у шарі «Heads»

Другий шар зображень – переіменований із «Head» у «Pinky Head», містить одне зображення і має шанс появи у вихідному зображенні лише один відсоток, це відображено на рисунку 3.11.

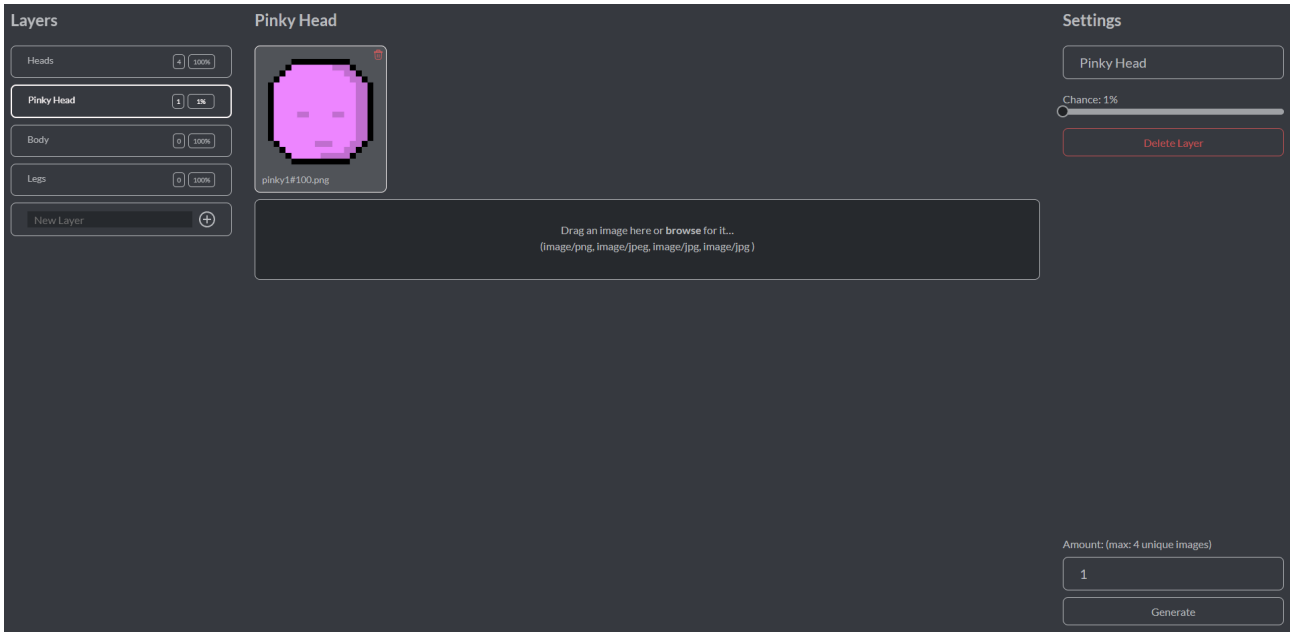

Рисунок 3.11 – Зображення у шарі «Pinky Head»

Третій шар зображень – переіменований із «Body» у «Hats», містить вісім зображенн і має шанс появи у вихідному зображенні у сорок відсотків, це відображено на рисунку 3.12.

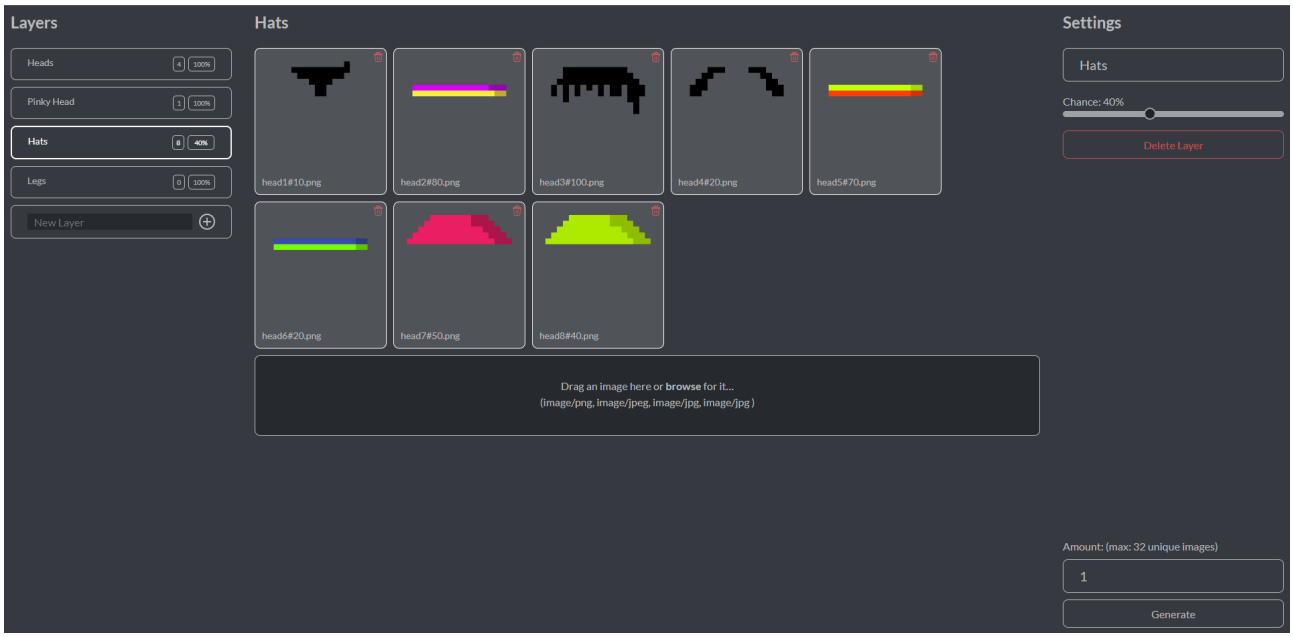

Рисунок 3.12 – Зображення у шарі «Hats»

Третій шар зображень – переіменований із «Legs» у «Faces», містить шість зображеннь і має стовідсотковий шанс появи у вихідному зображенні, це відображено на рисунку 3.13.

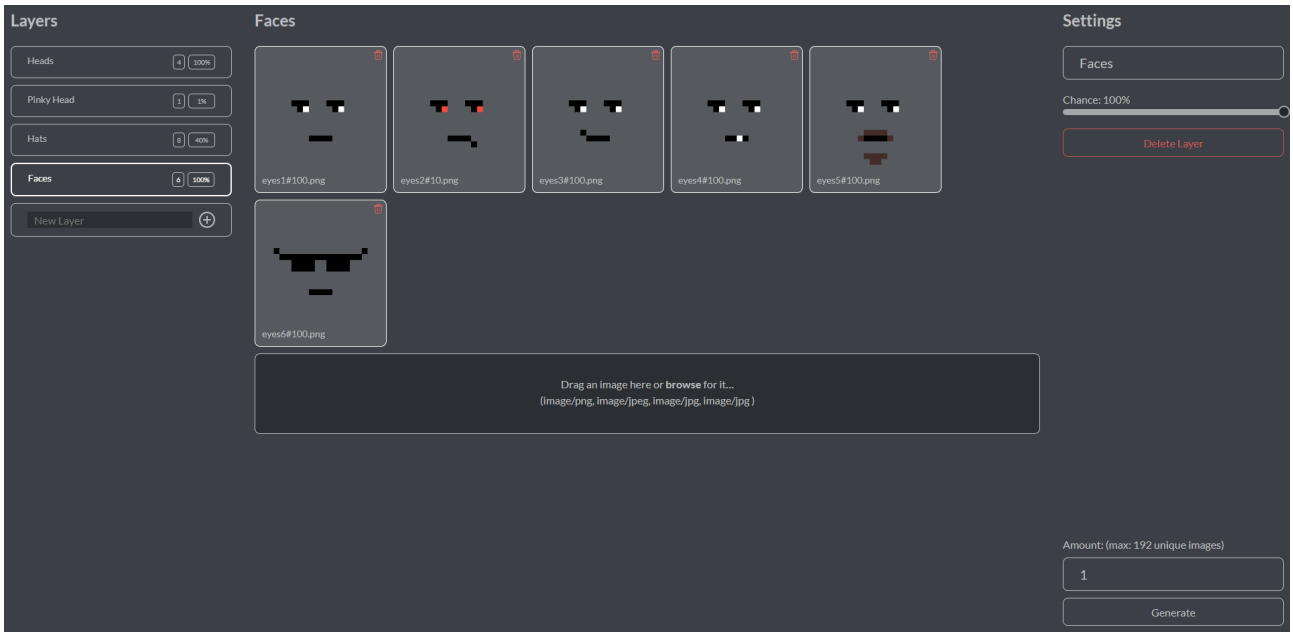

Рисунок 3.13 – Зображення у шарі «Faces»

Спробуємо згенерувати 102 зображення ввівши це значення до поля введення «Amount». Після натискання кнопки «Generate» динаміку генерації зображешь можна відслідкувати на кнопці генерації, це відображено на малюнках (3.14 – 3.15).

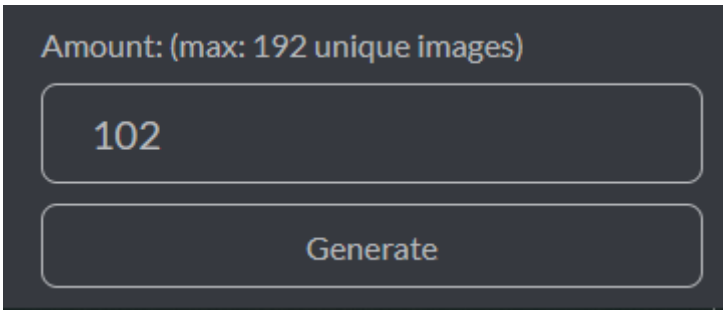

Рисунок 3.14 – Поле «Amount» та кнопка «Generate»

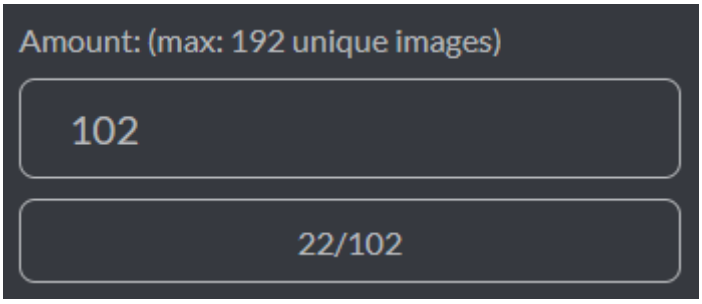

Рисунок 3.15 – Поле «Amount» та кнопка «Generate» із поточним станом генерації

Перед запуском додамо на початку основної функції генерації console.time(“test”), а на прикінці функції після створення архіву та початку його завантаження на пристрій користувача console.timeEnd(“test”) для виміру швидкості виконання функції генерації.

Після завершення генерації яка зайняла 1387.546875 мілісекунд, або ~1.4 секунди для 102 зображень до пристрою було завантажено архів з результатом генерації, у якому міститься 103 файли, з яких 102 згенерованих вихідних зображення та один файл «Traits.txt» що містить данні про сгенеровані зображення, вигляд архіву можна побачити на рисунку 3.16.

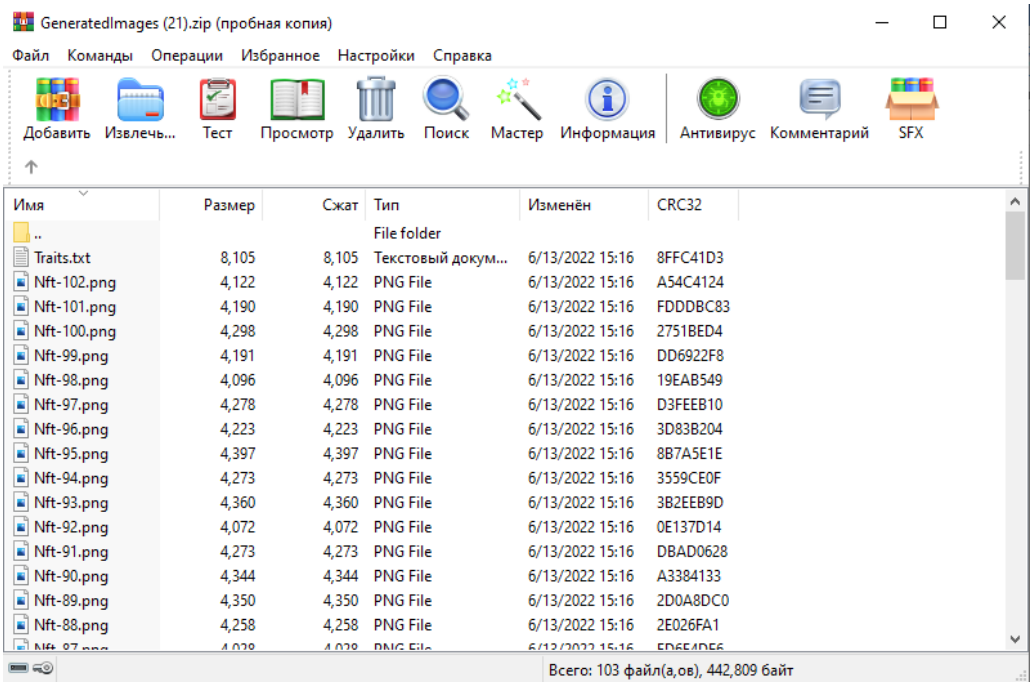

Рисунок 3.16 – Вигляд вихідного архіву

Разархівуємо його та розглянемо результат. Ми отримали 102 зображення лише два з яких отримали рідкий шар «Pinky Head» який мав шанс в один відсоток. Більше половини не мають шару «Hats» у якого був шанс сорок відсотків. Це можна побачити на рисунку 3.17. Зміст файлу «Traits.txt» зображено на рисунку 3.18.

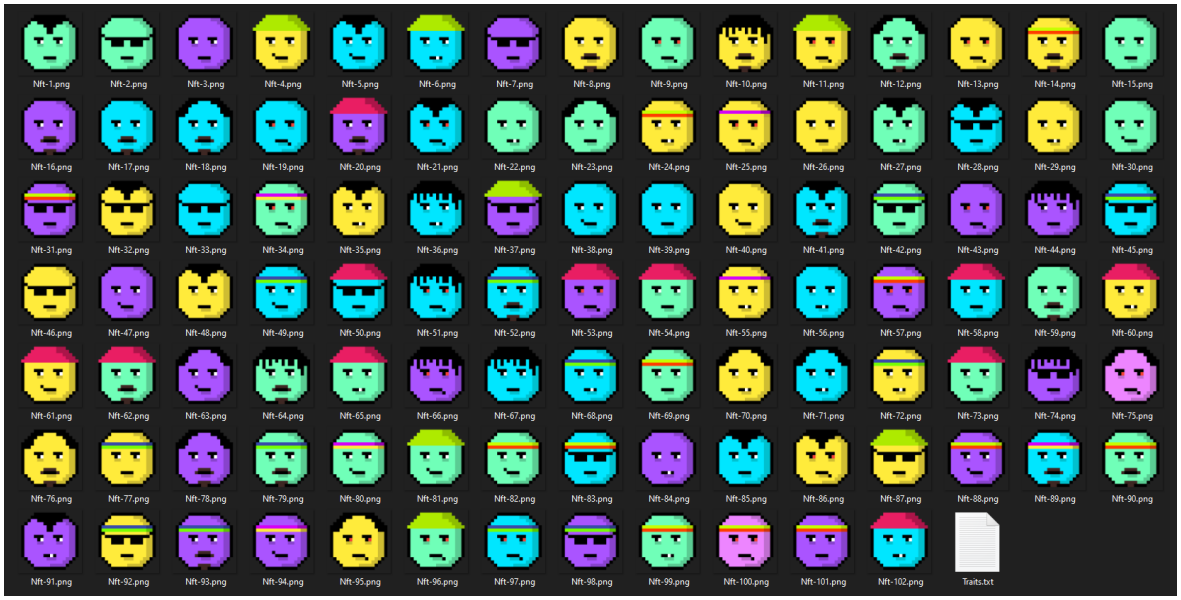

Рисунок 3.17 – Розархівований вихідний архів

Рисунок 3.18 – Вигляд вихідного архіву

3.6 Висновки за розділом

Було створено вебдодаток для генерації графічної колекції із наявних зображень користувача.

ВИСНОВОК

З початком роботи над дипломом були поставленні ряд певних задач, які успішно виконані:

- Проаналізовані та порівнянні аналоги розроблюємого вебдодатку, виделені їх переваги та недоліки;
- Проаналізовано та підібрані інструменти для розробки програмної частини вебдодатку дипломної роботи;
- Розроблено користувацький інтерфейс та функціональну частину додатку;
- Розроблено універсальний алгоритм генерації зображень;
- Проведено тестування програмного продукту;
- Написано пояснювальну записку дипломної роботи в відповідній структури з викладенням матеріалу в розділи.

Згодом, для вдосконалення вебдодатку для генерації графічних колекцій можуть бути додані наступні функції:

- Редагування шансів появи у вихідному зображенні для кожного окремого користувацького зображення;
- Демо режим із наданням користувачеві зображень для тестової генерації;
- Можливість змінювати порядок шарів зображень при генерації;
- Конструктор зображень та режим попереднього перегляду результатів роботи алгоритму генерації.

В результаті розроблений вебдодаток викладено на локальний хостинг та протестовано на працездатність. Також проведено тестування за різноплановими браузерами, якщо виникали помилки всі успішно виправлялися.

Отже результатом роботи являється повноцінний вебдодаток для генерації графічної колекції із наявних зображень користувача.

ПЕРЕЛІК ПОСИЛАНЬ

1. Nakamoto S. Bitcoin: A Peer-to-PeerElectronicCashSystem [Електронний ресурс] / SatoshiNakamoto. – 2008. – Режим доступу до ресурсу: https://bitcoin.org/bitcoin.pdf.

2. IBM [Електронний ресурс]. – Режим доступу: https://www.ibm.com/topics/smart-contracts.

3. Wikipedia [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Non-fungible_token.

4. Investopedia [Електронний ресурс]. – Режим доступу: https://www.investopedia.org/non-fungible-tokens-nft-5115211.

5. CyberScrilla [Електронний ресурс]. – Режим доступу: https://cyberscrilla.com/nft-smart-contracts-explained.

6. MDNWebDocs [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript.

7. JAVASCRIPT.INFO [Електронний ресурс]. – Режим доступу: https://javascript.info/intro.

8. SimpliLearn [Електронний ресурс]. – Режим доступу: https://simplilearn.com/tutorials/typescript-tutorial/typescript-vs-javascript.

9. Wikipedia [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/React_(JavaScript_library).

10. Wikipedia [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Visual_Studio_Code.

11. Wikipedia [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Npm_(software).

12.

ДОДАТОК А