

**UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ ÎN LIMBA
ROMÂNĂ**

LUCRARE DE LICENȚĂ

**Generarea text-conditionată a
imaginilor cu design-uri de tatuaje
folosind extinderea ariei lexicale a
textului input folosit în modelul
generativ**

**Conducător științific
Lect. MIRCEA Ioan Gabriel și Asist. ȚIRBAN Paul**

*Absolvent
Şapcă Ioana-Cătălina*

2022

ABSTRACT

This paper goes through the relevance, planning and implementation of a tattoo generating application.

The demand for tattoos has increased over the last years and people are running out of original ideas when it comes to tattoo designs. A tool that helps creating new, original tattoo designs could be well received by both tattoo artists and clients.

The image generation task has been studied in several scientific papers over the past 3 years. Currently, there are a few well known approaches for image generation: Generative Adversarial Networks, Vector Quantization Generative Adversarial Networks, Diffusion Models, Language Free Models etc. We went over the most significant papers for these approaches and discussed their benefits and drawbacks.

Generating images from open domain text prompts is a difficult undertaking that has necessitated the use of pricey and highly trained models. We present a methodology that uses expanded lexical fields as the input for several text-conditioned image generation instances and applies a method based on binary classification for ranking all of the candidate images to determine the best tattoo design. We are using a pretrained Vector Quantization Generative Adversarial Network among with the OpenAI's CLIP model for the image generation process (proposed in the paper [CBK⁺22]) and we demonstrate that the models combined with the semantic reformulation component and the ranking component are producing high quality images suitable as tattoo designs. Lastly, the final image is used to create two masked images, a *black and white image* and a *grey image*, both serving as a blueprint in order to ease the job of tattoo artists.

In order to encapsulate the tattoo generation model and it's features with a user-friendly application, we proposed a web client and a back-end server. We documented the application's components and use cases with detailed sequence diagrams.

We chose to implement the application's web client using ReactJS and the server-side in Java in order to ensure the scalability and the readability of the code. The decisions made were well documented and the implementation is though out to be easily deployed and extended.

We concluded that the proposed solution does improve the results as it brings the image to a closer representation of a tattoo. The application has a well-rounded user interface that is easy to use and pleasing to the eye. For a future iteration of the project, we could train the VQGAN models with a labeled dataset of tattoo images. This would result into more natural and conventional style tattoos. This, however, is outside the scope of this study because there is no such dataset available and it would have been extremely time consuming to collect and label all the necessary data.

Cuprins

1	Introducere	1
2	Motivația lucrării	3
2.1	Relevanța	4
3	Related work	5
3.1	Abordarea bazată pe modele generative	5
3.1.1	DTGAN - Dual Attention Generative Adversarial Networks pentru generarea text-conditionată de imagini	7
3.1.2	DM-GAN - Dynamic Memory Generative Adversarial Networks pentru generarea text-conditionată de imagini	8
3.1.3	AttnGAN - Folosirea unei Rețele Generative Adversare orientate spre detalii pentru generarea text-conditionată de imagini	9
3.2	Abordarea bazată pe Vector Quantization	9
3.2.1	Taming Transformers pentru generarea de imagini de înaltă rezoluție	12
3.2.2	AffectGAN - Generare de imagini conditionată de stări afective	12
3.2.3	VQGAN-CLIP: Generare și Editare de imagini îndrumate de limbajul natural	13
3.2.4	Metoda Zero-Shot pentru generare de imagini	14
3.2.5	CogView	14
3.3	Abordarea bazată pe modele de difuzie	14
3.3.1	GLIDE: Generarea și Editarea text-conditionată de imagini foto-realistice folosind Modele de Difuzie	15
3.4	Alte soluții relevante	15
3.4.1	CLIP	15
3.4.2	LAFITE : Generare de imagini folosind antrenare pe dataseturi fără etichete	16
3.5	Directii neexplorate	16
4	Soluția propusă	18
4.1	Specificații tehnice	18
4.2	FlowChart-uri pentru soluția propusă	20
4.3	Implementarea soluției	20
4.4	Specificarea și testarea API-ului	21
4.5	Rezultatele generării	23
4.6	Evaluarea performantei API-ului propus	23

5 Aplicatie	28
5.1 Aplicația propusă	28
5.2 Arhitectura aplicatiei	28
5.2.1 Comportament - cazuri de utilizare	29
5.2.2 Structura - diagrama de componente	29
5.2.3 Interacțiuni ale componentelor în cazurile de utilizare	31
5.3 Design Patterns	37
6 Implementarea aplicatiei	39
6.1 Implementarea arhitecturii abstracte	39
6.2 Limbajele de programare și tehnologiile folosite	39
6.2.1 Serverul aplicatiei	40
6.2.2 Serverul de generare de imagini	41
6.2.3 Clientul Web	41
6.2.4 Specificații endpoint-uri	42
6.2.5 Testarea aplicatiei	49
6.3 Detalii de implementare	50
6.4 Persistența datelor	50
6.5 UX și manual de utilizare	52
7 Concluzii și dezvoltări ulterioare	58
7.1 Dezvoltări ulterioare	58
Bibliografie	59

Capitolul 1

Introducere

Generarea de imagini cu design-uri de tatuaje pornind de la un text input este o sarcină dificilă care necesită antrenarea unui model pe reprezentări vizuale și textuale. Manipularea datelor de tip perechi text-imagine este o problemă abordată în mai multe feluri de o multitudine de lucrări științifice.

În această lucrare propun o soluție pentru generarea de tatuaje folosind un model generativ îmbunătățit cu o componentă de extindere a spațiului lexical al descrierii input și o componentă de ranking a imaginilor candidat. Componenta de extindere lexicală constă în reformularea descrierii primite de la utilizator, astfel încât să se obțină mai multe descrieri echivalente semantic, dar diferite lexical. Componenta de ranking constă în notarea mai multor imagini candidat pe baza mai multor clasificări binare efectuate cu ajutorul modelului CLIP [RKH⁺21]. Pentru modelul generativ se folosește una din metodele State-of-The-Art în generarea de imagini 3. Pentru ca această funcționalitate să fie ușor accesibilă de către public, a fost construită o aplicație web prin care utilizatorul își poate genera unul sau mai multe tatuaje. Ulterior, lista de tatuaje poate fi accesată și gestionată de către utilizator prin intermediul contului creat pe platformă. Aplicația este segmentată în 3 componente: componenta de generare de tatuaje (un server Python care pune la dispoziție această funcționalitate de generare printr-un API), componenta de front-end (punctul de interacțiune al utilizatorului cu platforma), componenta de back-end (responsabilă de persistența datelor și de logica interacțiunii dintre componenta de front-end și cea de generare de tatuaje).

Lucrarea aduce următoarele contribuții:

1. Adăugarea unei componente de extindere a ariei lexicale a textului input folosit de modelul generativ. Un avantaj semnificant al acestei componente este acela că dezvoltarea lexicală a textului input duce la o rază mai mare de acoperire a claselor de entități învățate de modelul generativ folosit, indiferent de natura acestuia. În consecință, sansele generării unei imagini concludente și fidele cu descrierea initială sunt mai mari.
2. Adăugarea unui modul de ranking al imaginilor candidat bazat pe clasificări binare. Această componentă face ca imaginea finală să fie reprezentativă unui design de tatuaj.

3. Adăugarea unei componente de aplicare de filtre peste imaginea finală pentru crearea blueprint-ului tatuajului. Din imaginea finală sunt obținute alte două imagini: o imagine cu filtru *alb-negru* și o imagine cu filtru *gri*.
4. Crearea unei aplicații web ce încapsulează funcționalitatea principală a lucrării (generarea de tatuaje) și oferă modalități de vizualizare și gestionare a tatuajelor.

Capitolul 2

Motivația lucrării

De-a lungul timpului inteligența artificială a înlocuit munca oamenilor în fabrici, magazine și depozite. Cu cât tehnologia și puterea de computație avansează, cu atât domeniile în care putem beneficia de automatizare se extind. Începem să vedem din ce în ce mai multe poziții pe care nu mai este nevoie de munca oamenilor, iar task-urile sunt preluate de softuri ghidate de inteligență artificială.

Încă din anii '60 oamenii au început să fie interesati de generarea de artă folosind calculatoare digitale [RNo66]. La început aceste imagini erau compuse doar din linii verticale și orizontale 2.2, asemănător compozițiilor pictorului francez Piet Mondrian. Cu toate acestea, imaginile erau apreciate, ba chiar considerate mai frumoase și mai inedite decât cele reale 2.1.

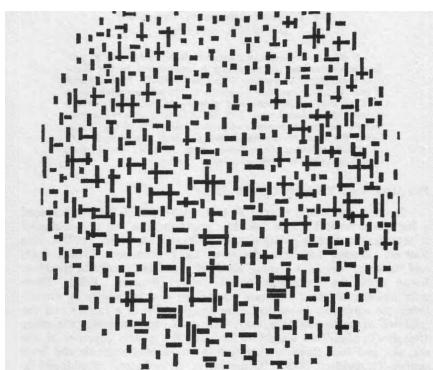


Figura 2.1: "Composition With Lines" (1917) by Piet Mondrian.

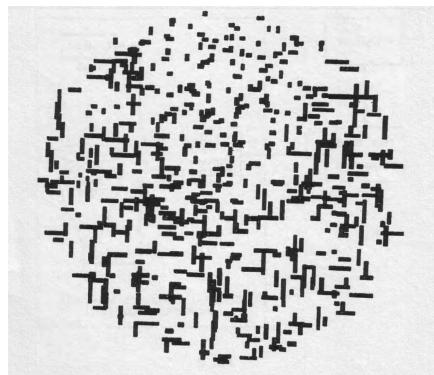


Figura 2.2: "Computer Composition With Lines" (1964) in association with an IBM 7094 digital computer

În ultimii ani interesul pentru arta generată digital a crescut foarte mult, datorită capabilităților noilor algoritmi de machine learning și a componentelor foarte eficiente (procesoare și plăci video). Mai exact, în ultimii 2 ani au fost explorate mai multe metode de generare de imagini, însă acest concept este încă într-o stare incipientă.

2.1 Relevanță

Alegerea unui tatuaj este o decizie importantă pentru o persoană care își dorește acest lucru. Datorită naturii permanente a unui tatuaj, designul, forma, locul și dimensiunea acestuia sunt factori cheie care influențează foarte mult satisfacția clientului. În majoritatea cazurilor, artistul tatuator ia rolul de consultant al clientului, oferind idei, propuneri și expertiza în domeniu. Cu toate acestea, proiectarea și stilizarea design-ului tatuajului dorit de client nu se află printre calificările tuturor artiștilor tatuatori [Bar13].

O mare parte din oamenii care doresc să își facă un tatuaj, cunosc o listă de elemente/componente pe care și le doresc a fi integrate în tatuaj, însă apelează la artistul tatuator pentru a le oferi imagini de referință și propuneri de design. Deoarece este greu să găsești o imagine pe placul clientului care să conțină toate elementele cerute de acesta, se recurge la căutarea mai multor imagini de referință care sunt folosite pentru crearea design-ului final. Un software care generează imagini pe baza unei descrieri ar fi o soluție care ar revoluționa tot acest proces și ar ajuta mult la luarea unei decizii, chiar înainte de a intra într-un salon.

O altă problemă des întâlnită este aceea că multe persoane aleg tatuaje similare și repetitive. Aici putem aminti de tatuaje cu elemente precum busole, ancore, trandafiri etc. Folosindu-ne de computer vision în generarea de imagini pentru tatuaje am putea crește diversitatea.

Pentru că o imagine face cât 1000 de cuvinte, ar ajuta ca în generarea unui tatuaj, clientul să se folosească atât de un text, cât și de o imagine de start. Studii arată că tot mai multe persoane apelează la tratamente de înlăturare a unui tatuaj vechi, iar un mare procentaj dintre aceste persoane o fac datorită design-ului prost și a nemultumirii. În aceste cazuri, preluarea imaginii vechiului tatuaj și a descrierii dorite de client ar putea duce la înnoirea și îmbunătățirea vechiului tatuaj în locul stergerii acestuia.

Capitolul 3

Related work

Pentru că problema generării de tatuaje este o particularitate a generării text-conditionată de imagini, se va extinde problema astfel.

În prezent există mai multe modele care abordează problema generării imaginilor pe baza de text. Majoritatea soluțiilor State-of-The-Art folosesc rețele generative adversare, însă există și abordări diferite. În secțiunile care urmează sunt prezentate cele mai importante lucrări științifice din acest domeniu, incluzând punctele slabe, punctele forte și rezultatele experimentelor lor în comparație cu celelalte metode State-of-The-Art.

Ca și metode de evaluare ale unui algoritm de generare de imagini se folosesc următoarele metriki:

Inception Score (IS) Măsoară cât de realistic și variat arată imaginile generate. Funcția returnează un scor pentru un set de imagini [BS18].

Fréchet Inception Distance (FID) Măsoară diferența dintre distribuția unei imagini generate și a unei imagini reale folosite pentru antrenarea generatorului [YZD21].

3.1 Abordarea bazată pe modele generative

Principalul dezavantaj al metodelor generative bazate pe embedding este acela că nu se pot plia pe schimbări de domeniu. Acest lucru înseamnă că odată antrenat, modelul va putea prezice doar imagini cu elemente din clasele prezente în etapa de învățare. De asemenea, metoda nu garantează că funcția de mapare va funcționa corespunzător pentru imagini care nu aparțin domeniului cunoscut de model. Pentru ca acest dezavantaj să poată fi depășit, este important ca modelul să fie antrenat atât pe imagini din categorii cunoscute, cât și pe imagini din categorii necunoscute.

Deoarece în problema generării de imagini este costisitor să antrenezi modelul cu extrem de multe clase, este importantă explorarea modelelor cu învățare de tip zero-shot. Învățarea zero-shot constă în antrenarea unui model astfel încât

acesta să devină capabil să distingă elemente din clase pe care nu le cunoaște. În general, acest lucru este obținut folosind o rețea de tipul Conditional Generative Adversarial Network (cGAN) care generează imagini condiționate de unul sau mai multe atrbute semantice (text).

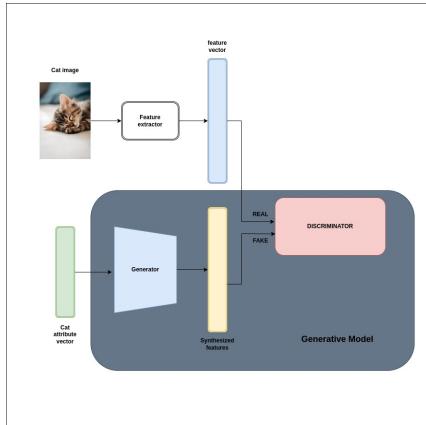


Figura 3.1: Model generativ bazat pe învățare zero-shot

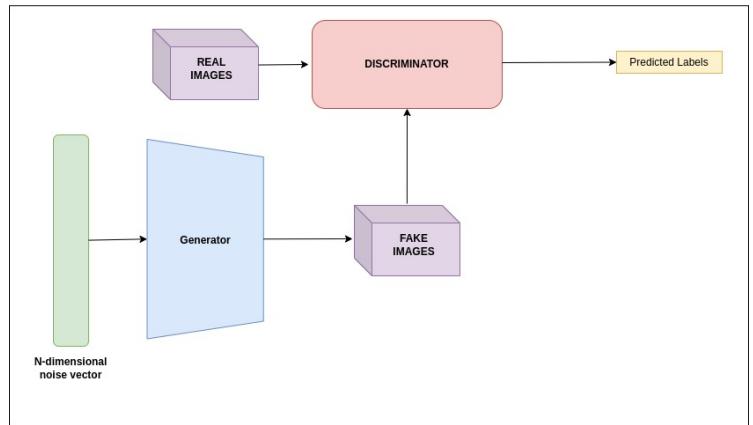


Figura 3.2: Procesul de predictie al unui algoritm GAN

Diagrama 3.1 exemplifică modul de funcționare al unui algoritm generativ bazat pe învățare zero-shot. Într-un astfel de model, este extras vectorul de features din fiecare imagine de antrenament, iar vectorul de atrbute corespunzător imaginii este trecut prin rețea generatoare pentru a se obține un vector sintetizat de features. Este important ca vectorul sintetizat de features să reprezinte cât mai fidel vectorul original de features. În faza de antrenament sunt observate imagini atât din dataset-ul de training, cât și date aditionale (învățare zero-shot).

În modelele adverse de generare se adaugă o componentă care evaluează output-ul generatorului. Aceasta, discriminatorul încearcă să distingă datele de antrenament de imaginile generate de generator (evaluarea autenticității), iar generatorul încearcă să convingă discriminatorul că imaginile generate de el sunt reale. Astfel, generatorul învăță să genereze imagini din ce în ce mai bune și discriminatorul devine tot mai precis în clasificare. Cu alte cuvinte, cele două rețele concurează una împotriva celeilalte.

Este important ca în faza de antrenament a discriminatorului, valorile generatorului să fie constante, iar în faza de antrenament a generatorului, valorile discriminatorului să fie constante pentru ca nivelurile de calificare ale celor două rețele să fie asemănătoare. În caz contrar, generatorul va exploata punctele slabe ale discriminantului ceea ce ar duce la returnarea de valori false negative.

Pașii pentru antrenarea unui Generative Adversarial Network sunt:

- Definirea problemei
- Alegerea arhitecturii de GAN
- Antrenarea discriminatorului pe date reale
- Generarea de date input false pentru generator

- Antrenarea discriminatorului pe date false
- Antrenarea generatorului cu datele output de la discriminator

3.1.1 DTGAN - Dual Attention Generative Adversarial Networks pentru generarea text-condiționată de imagini

Soluția [ZS21] propune o rețea generativă adversară numită DT-GAN (Dual Attention Generative Adversarial Network) care sintetizează imagini de înaltă rezoluție și fidele semantic cu textul primit, folosind o singură pereche generator-discriminator. Majoritatea soluțiilor State-of-The-Art folosesc și antrenează o serie de mai mulți generatori și discriminatori; în fiecare etapă generatorul preluând rezultatul obținut de generatorul precedent și folosindu-l ca dată de input. Creșterea numărului de rețele crește proporțional timpul de computație necesar antrenării acestora. În plus, rețelele generatoare trebuie antrenate în ordine (ultimul generator nu poate fi antrenat până când toate celelalte generatoare precedente lui nu ajung la optimul global).

Modelul propus este compus din patru componente: două module de atenție (un channel-aware attention module și un pixel-aware attention module), un layer de normalizare (Conditional Adaptive Instance-Layer Normalization - CAdaILN) și o componentă de visual loss. Cele două module de atenție sunt folosite pentru a ghida generatorul să se "concentreze" pe pixelii și channel-urile relevante și să ignore pixelii și channel-urile irelevante. Funcția de normalizare se aplică pe fiecare imagine generată în parte, spre deosebire de alte soluții care aplică Batch Normalization (normalizare pe un set de imagini). Această componentă de Conditional Adaptive Instance-Layer Normalization permite o mai bună controlare a schimbărilor de formă și textură (comparativ cu descrierea input). Componenta de Visual Loss crește aspectul de imagine reală a imaginii sintetizate prin calcularea similarității distribuțiilor de formă și culoare dintre imaginea reală și cea generată (în etapa de antrenare).

În figura 3.3 se poate observa interacțiunea dintre aceste componente.

Comparativ cu alte soluții State-of-The-Art în generarea de imagini bazate pe Generative Adversarial Networks, cum sunt StackGAN++ [ZXL⁺17] și Attn-GAN [XZH⁺17], DTGAN [ZPCY19] (împreună cu cele 4 componente menționate) a obținut cel mai bun Inception Score (4.88) pe dataset-ul CUB și cel mai bun Fréchet Inception Distance (16.35 și 23.61) pe dataset-urile CUB, respectiv COCO. Din punct de vedere calitativ, această soluție realizează imagini detaliate și clare pentru clasele pe care este antrenat, însă nu are capabilități de generare zero-shot și nu dă rezultate bune pentru descrieri complexe și neobișnuite.

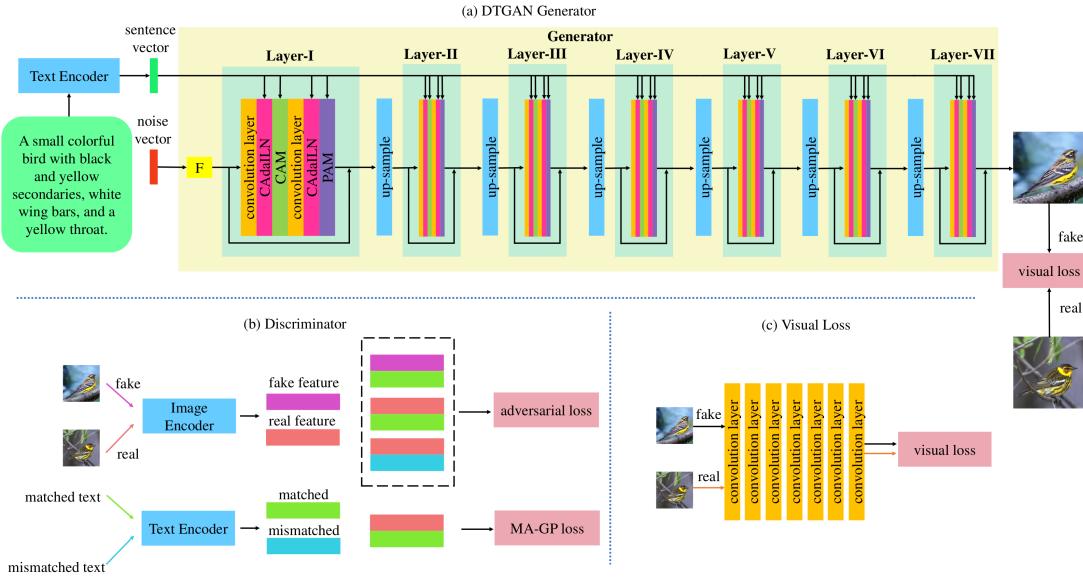


Figura 3.3: Flow-ul antrenării rețelei DTGAN [ZS21]

3.1.2 DM-GAN - Dynamic Memory Generative Adversarial Networks pentru generarea text-conditionată de imagini

Această soluție încearcă să rezolve câteva dintre minusurile algoritmilor generativi. În primul rând, rezultatele generării depind foarte mult de primele generații de imagini. Astfel, imaginile necesită, în general, o etapă de rafinare care împiedică rezultatul final să aibă o rezoluție mare. În al doilea rând, majoritatea celorlalte modele generative utilizează același mod de reprezentare a cuvintelor în toate procesele de rafinare ale imaginilor. Dynamic Memory Generative Adversarial Network [ZPCY19] propune un mecanism de selectare a cuvintelor cheie, relevante pentru imaginea finală cu ajutorul unui memory gate, dar și un mecanism de redresare a imaginilor generate greșit pentru creșterea calității imaginii finale.

În mod practic, pornind de la o descriere, soluția generează inițial o imagine primitivă, blurată și fără detalii. Apoi, această imagine este pasată în etapa de rafinare 3.4. Procesul de rafinare poate fi repetat de mai multe ori pentru un rezultat mai calitativ.

În urma evaluării pe dataset-urile CUB și COCO, soluția DM-GAN a obținut un Inception Score de 4.75, respectiv 30.49 și un Fréchet Inception Distance de 16.09, respectiv 32.64, ceea ce placează soluția pe locul 2 după DT-GAN. Experimentele au fost făcute în comparație cu modelele StackGAN, AttnGAN și PPGN [YWL⁺20], DT-GAN și GAN-INT-CLS [RAY⁺16]. Din punct de vedere calitativ, modelul generează imagini de o rezoluție vizibil mai mare decât StackGAN [ZXL⁺16] și AttnGAN[XZH⁺17], însă imaginile nu sunt întotdeauna realiste.

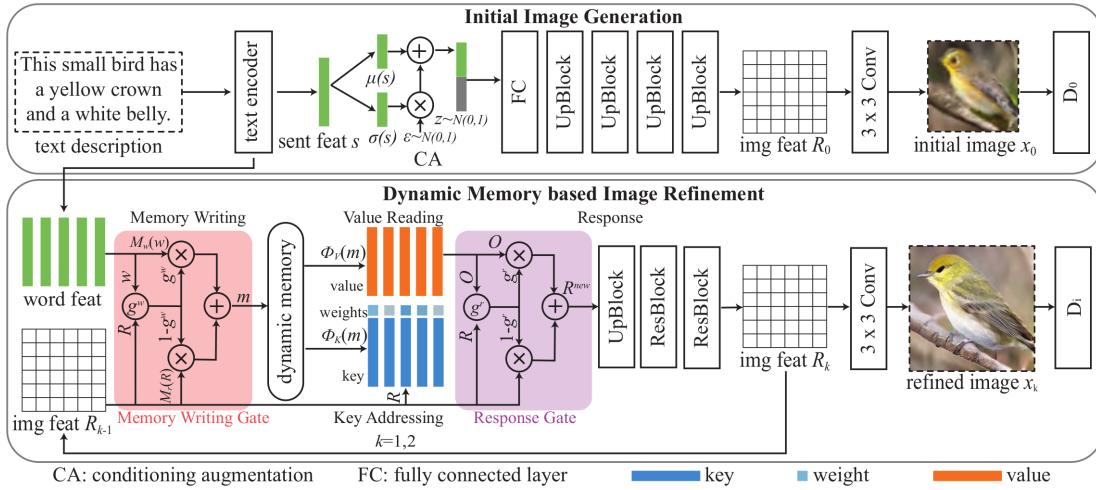


Figura 3.4: Arhitectura modelului DM-GAN [ZPCY19]

3.1.3 AttnGAN - Folosirea unei Rețele Generative Adversare orientate spre detalii pentru generarea text-conditionată de imagini

În lucrarea științifică [XZH⁺17] se propune un Attentional Generative Adversarial Network (AttnGAN) care permite o generare a imaginilor orientată spre detalii. Această generare se face pe etape, ținându-se cont de cuvintele cheie din descrierea primită. Modelul se constituie din două componente: un Attentional Generative Network (o rețea de mai mulți generatori) și un Deep Attentional Multimodal Similarity Model (două rețele neuronale de mapare a subregiunilor din imagini în elemente semantice).

Experimentele făcute pe dataset-urile CUB și COCO arată că AttnGAN obține un Inception Score de 4.36, respectiv 25.89, ceea ce nu întrece modelele abordate anterior: DT-GAN și DM-GAN. Totodată, soluția nu generează imagini realiste nici pentru texte input simple și are nevoie de antrenamentul mai multor rețele neuronale decât soluțiile anterioare.

Există și alte modele și soluții propuse precum [Mul], [Imp], [RZZ⁺21] și altele care însă nu obțin rezultate concluzante și nu au fost folosite ca referință în elaborarea și dezvoltarea soluției propuse în această lucrare.

3.2 Abordarea bazată pe Vector Quantization

Vector Quantized Generative Adversarial Network (VQGAN) este o variație a modelului Vector Quantized Variational Autoencoder (VQVAE) care integrează arhitectura unui Generative Adversarial Network (GAN).

Un autoencoder este un model de compresie a datelor, în special al imaginilor. Autoencoderul preia o imagine ca data de input și o transpune într-un spațiu la-

tent (low-dimension latent space), apoi este preluată de decoder și transformată înapoi în spațiul imaginii. Astfel imaginea reconstruită este foarte asemănătoare cu imaginea inițială. Diferența dintre imaginea finală și cea inițială se calculează ca fiind loss-ul reconstrucției. Se numește un model de autoencodare bun, un model care produce o pierdere cât mai mică de informație. Pentru ca modelul să piardă cât mai puțină informație în procesul de encodare și decodare, acesta este antrenat prin gradient descent.

În spațiul latent al unui autoencoder, imaginile similare nu se situează în locuri apropiate [3.5]. Deci, două puncte apropiate din spațiul latent pot fi decodificate ca imagini foarte diferite. Cu alte cuvinte, funcția de mapare/encodare nu tine cont de similaritățile datelor/imaginilor în generarea vectorilor latenti și acest lucru împiedică folosirea unui astfel de autoencoder într-un proces generativ.

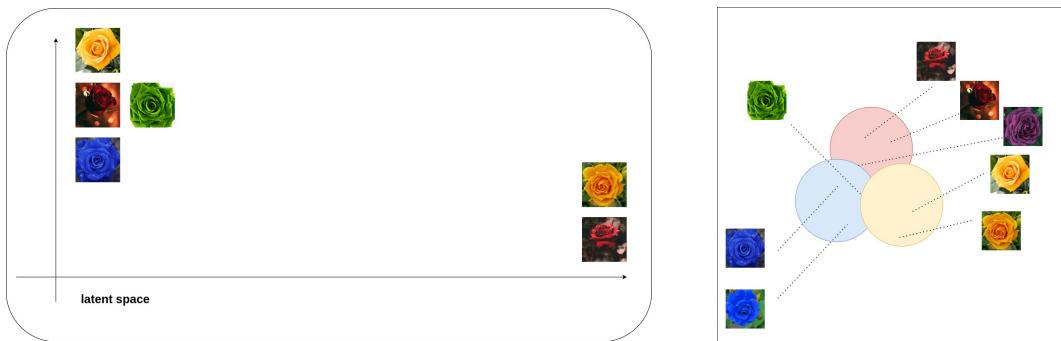


Figura 3.5: Autoencoder image mapping in the latent space

Figura 3.6: Regularisation distribution of VAE - for reference purpose only

Variational Autoencoder (VAE) reparametrizează spațiul latent astfel încât vectorii latenti ai imaginilor asemănătoare să fie situați în apropiere [3.6], rezolvând problema autoencoderelor originale. VAE returnează o distribuție în spațiul latent, în locul unui singur punct și adaugă un termen de regularizare în funcția de loss astfel încât distribuția returnată să asigure o regularitate/ordonare a datelor în spațiul latent.

VQVAE folosește arhitectura de Variational Autoencoder, adăugând componenta de Vector Quantization care permite învățarea unui set de reguli pentru generarea de imagini. Vector Quantization (VQ) se folosește de un Codebook pentru a înlocui fiecare vector latent din reprezentarea imaginii inițiale cu cel mai apropiat vector din Codebook [3.7]. Astfel, în urma decodificării, se va obține imaginea reconstruită. Modelul de Vector Quantization învăță acest CodeBook în faza de antrenament, astfel că se vor crea pattern-uri în Codebook pentru elemente regăsite în dataset-ul de training (exemplu: fundal verde, floare, câine etc.). Codebook-ul devine o unealtă de construire a imaginilor în etapa de generare.

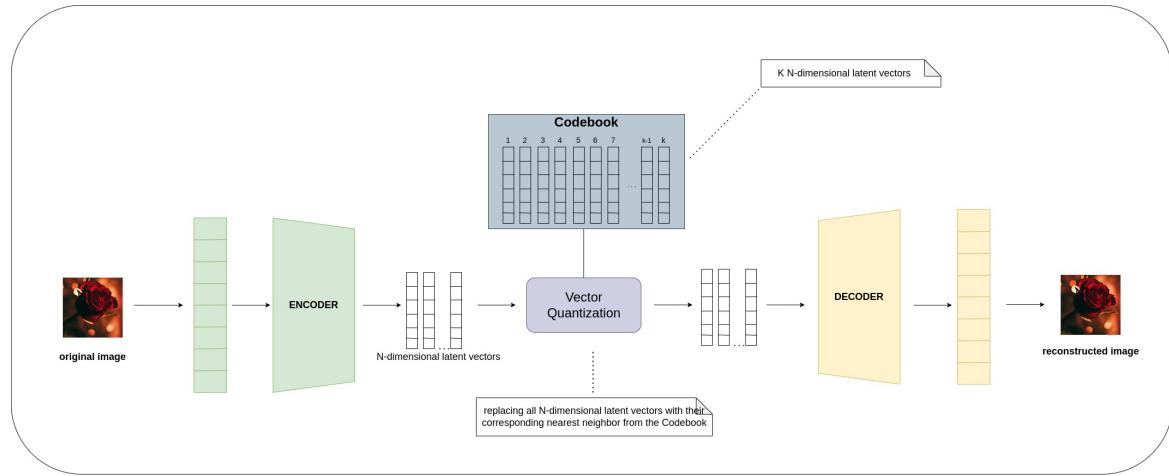


Figura 3.7: Vector Quantization Variational Autoencoder Workflow - diagram created based on [ERO20]

Vector Quantization Generative Adversarial Network (VQGAN) 3.11 folosește arhitectura de Generative Adversarial Network, deci folosește o rețea de tip Generator (componenta VQVAE) și o rețea de tip Discriminator. Discriminatorul primește ca input în faza de antrenare atât imagini din dataset-ul de training, cât și imagini reconstruite pentru a decide dacă acestea sunt reale sau nu. Discriminatorul ajută Autoencoderul să producă imagini tot mai realistice, iar Autoencoderul ajută Discriminatorul să devină mai exact în prezicerile sale.

În etapa de generare este nevoie de o componentă numită Transformer care primește un input de tip text și returnează vectori latenti care pot fi folosiți de către Vector Quantizator.

Imaginiile rezultate în urma generării pot fi mai mari decât dimensiunea vectorilor din spațiul latent sau decât dimensiunea imaginilor de training. Atât Encoderul cât și Decoderul folosesc arhitectura convolutională, ceea ce permite Transformerului să trimite mai mulți vectori latenti către Decoder pentru a reconstrui o imagine de o dimensiune mai mare.

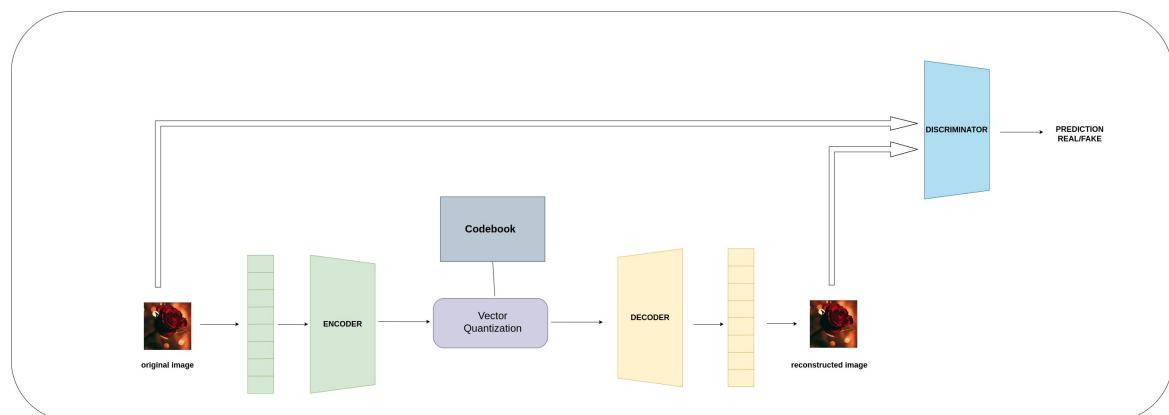


Figura 3.8: Vector Quantization Generative Adversarial Network (VQ-GAN) Workflow - diagram created based on [ERO20]

3.2.1 Taming Transformers pentru generarea de imagini de înaltă rezoluție

Această metodă [ERO20] folosește arhitectura convolutională de Vector Quantization Generative Adversarial Network pentru generarea de imagini pe bază de descriere. Este prima soluție care generează imagini de dimensiuni foarte mari. Pentru generarea de imagini 256x256, folosindu-se dataset-ul COCO, această soluție a obținut al treilea cel mai mic FID, comparativ cu soluțiile StyleGAN2, U-Net GAN, BigGAN și altele. Din perspectiva calitativă, atât imaginile de dimensiuni mici (256x256 pixels), cât și imaginile de dimensiuni mari (1280x832 pixels, 1024x416 pixels, 1280x240 pixels) sunt detaliate și foarte realistice.

3.2.2 AffectGAN - Generare de imagini condiționată de stări afective

AffectGAN [GLY21] generează imagini artistice care exprimă anumite stări afective. Soluția folosește un Generative Adversarial Network pentru deep learning, modelul semantic CLIP [RKH⁺21] oferit de OpenAI și dataset-ul WikiArt. Solutia este construită pe două modele: modelul CLIP și modelul VQGAN. Modelul CLIP este folosit pentru a introduce informațiile semantice (text input) în procesul de generare. VQGAN (Vector Quantized Generative Adversarial Network) a fost antrenat pe dataset-ul WikiArt care conține fotografii artistice și tablouri.

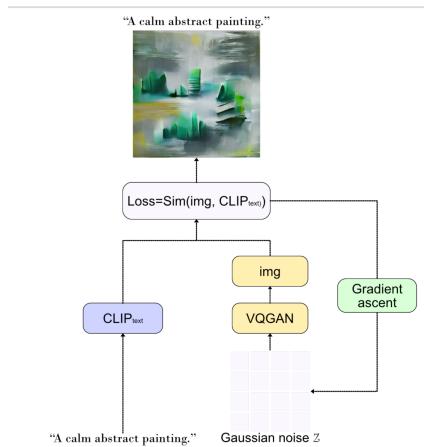


Figura 3.9: AffectGAN Image Generation Workflow [GLY21]

Modelul VQGAN generează câte o posibilă imagine rezultat; această imagine este comparată cu text embedding-ul obținut cu ajutorul modelului CLIP și se obține valoarea loss-ului. Loss-ul este minimizat prin intermediul optimizatorului AdamW pentru a se obține o imagine cât mai concludentă (3.9).

Evaluarea corectitudinii rezultatelor a fost făcută pe un eșantion de 50 de persoane care au estimat acest model la 2.95/5 din punct de vedere al calității imaginii și calității clasificării emoțiilor. Lucrarea nu oferă scorurile de Inception și FID. Rezultatele acestei soluții pot fi îmbunătățire prin label-uirea mai concludentă a imaginilor din dataset.

3.2.3 VQGAN-CLIP: Generare și Editare de imagini îndrumate de limbajul natural

Articolul [CBK⁺22] propune o soluție care permite generarea și editarea imaginilor pe baza unui input text. Metoda folosește modelul CLIP pentru a ajuta o rețea de tip Vector Quantization Generative Adversarial Network (VQGAN) și demonstrează că această variantă produce imagini mai calitative decât minDALL-E, GLIDE și Open-Edit.

Se pornește de la textul input și se folosește un GAN pentru generarea iterativă de imagini candidat. La fiecare pas se folosește modelul CLIP pentru îmbunătățirea imaginii astfel: se calculează loss-ul ca fiind diferența sferică pătrată dintre embedding-ul imaginii generate de GAN și embedding-ul textului input.

Pentru generare, imaginea de start conține pixeli generați aleatoriu. Procesul de optimizare este reluat până când imaginea rezultată se potrivește cu descrierea inițială (textul input). Cu alte cuvinte, procesul de optimizare are loc până când loss-ul calculat cu ajutorul CLIP este sub un anumit prag. Pentru editare procesul este similar, cu mențiunea că imaginea de start este primită ca și input. Altfel, diferența dintre a folosi această tehnică pentru generare sau editare de imagini este modul în care este initializat generatorul: cu un random noise (pentru generare) sau cu o imagine (pentru editare).

Schimbările de la o iteratie la alta sunt puternice dacă loss-ul este calculat pentru o singură imagine. Pentru a redresa acest lucru, din imaginea generată la fiecare pas sunt generate mai multe imagini augmentate pe care se aplică tehnici de rotire, oglindire, noising etc, iar mai apoi este calculat loss-ul prin intermediul CLIP. Detaliile semantice semnificative nu sunt pierdute în acest proces, iar media loss-ului reduce variațiile mari de la un pas la altul al generării.

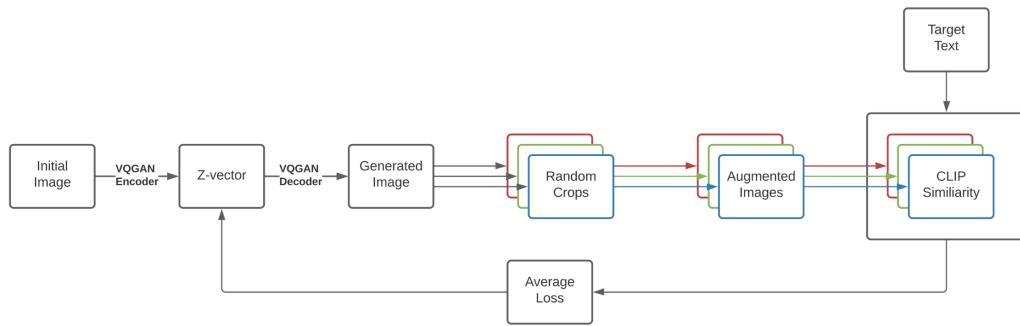


Figura 3.10: VQGAN + CLIP image optimization [CBK⁺22]

Metoda permite generări de imagini într-un anumit stil specificat în textul input (exemplu: "an astronaut in the style of van Gogh").

Marele avantaj prezentat de această tehnică este faptul că modelul nu trebuie antrenat în plus pentru editarea imaginilor, este suficient antrenamentul făcut pentru generare (antrenarea generatorului și al unui encoder image-text). De asemenea, imaginile generate respectă fidel textul input chiar și atunci când

textul input descrie o scenă ireală și improbabilă (exemplu de text input ireal: "un câine în tutu jucând sah").

Modelul oferă posibilitatea editării imaginilor prin schimbarea paletei cromatice fără denaturarea detaliilor, spre deosebire de modelul Open-Edit. Printre capabilitățile editării imaginilor se numără și schimbarea vremii, clarității și a focusului.

Din punct de vedere cantitativ, soluția nu prezintă comparații și rezultate. Din punct de vedere calitativ, soluția generează imagini impresionante și fidele cu descrierea semantică în majoritatea cazurilor, cu excepția fețelor.

3.2.4 Metoda Zero-Shot pentru generare de imagini

Această soluție [RPG⁺21] este cunoscută sub numele de DALL-E și este constituită din două componente. Prima componentă, cea de Vector Quantization Variational Autoencoder (VQ-VAE) are rolul de a encoda, decoda și compresa imaginile. Cea de-a doua componentă este un Autoregressive Transformer antrenat pe spațiul latent creat de componenta VQVAE.

Soluția generează imagini de mici dimensiuni și calitatea acestora este între-cută de cea a altor modele precum DM-GAN și Validation.

3.2.5 CogView

CogView [DYH⁺21] este un Text-to-Image Generation Model care se folosește de un Transformer cu 4 miliarde de parametri și de modelul Vector Quantization Variational Autoencoder (VQ-VAE).

Calitativ, soluția dă rezultate foarte bune pentru generarea de articole de modă și pentru generarea de imagini într-un anumit stil precizat (exemplu: sketch, cartoon, painting). CogView obține un scor IS de 18.2 (locul 4/5 - unde locul 1 are cel mai bun scor) și un scor FID (locul 2/5 - unde locul 1 are cel mai bun scor) comparativ cu modelele DALL-E, DF-GAN, DM-GAN, AttnGAN. Deoarece modelul este proiectat să lucreze cu descrieri în limba chineză, experimentele au fost făcute pe un eșantion de 30.000 de imagini din dataset-ul MS COCO pentru care descrierile au fost traduse în limba chineză.

3.3 Abordarea bazată pe modele de difuzie

Modelul de difuzie este o derivare din modelul generativ. Modelul de difuzie sau Diffusion Model transformă fiecare imagine din dataset-ul de training cu ajutorul unei funcții de difuzie. Cu alte cuvinte, se pornește un proces iterativ în care unei imagini îi este adăugat un noise gaussian foarte mic, treptat, până când această imagine devine complet difuzată. Se dorește antrenarea modelului astfel încât plecând de la o imagine x difuză să se poată prezice imaginea y anterioară acesteia. În practică, se antrenează modelul pentru prezicerea cât mai exactă a noise-ului/zgomotului dintre două imagini x și y date.

Pentru generarea de imagini, se pleacă de la un sample din data distribution și se reface procesul de difuzie prin care modelul prezice următorul termen din distribuție. Următorul termen din distribuție este calculat adăugându-se valoarea zgomotului Gaussian (Gaussian noise) la termenul curent.

Modelele de difuzie au dat rezultate impresionante în domeniul generării, manipulării și modificării imaginilor.

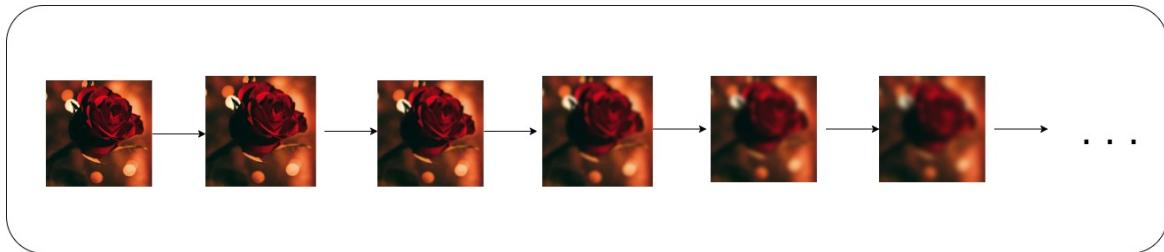


Figura 3.11: Image Diffusion Process - for reference purpose only

3.3.1 GLIDE: Generarea și Editarea text-conditionată de imagini foto-realisticе folosind Modele de Difuzie

Soluția [NDR⁺21] explorează modelele de difuzie pentru generarea de imagini de înaltă calitate. Sunt examineate două strategii de îndrumare: îndrumare CLIP și îndrumare fără clasificare. Atât CLIP cât și modelul classifier-free sunt folosite pentru calcularea loss-ului dintre imaginea generată și textul input.

Modelul este capabil să facă editări realistice ale unei imagini existente prin selectarea ariei care se dorește a fi modificată și precizarea modificării dorite printr-o descriere. Editarea de imagine se face similar generării, cu diferența că imaginea de start este cea input.

Pentru experimente de generare de imagini text-conditionate a fost antrenat un model de difuzie cu 3.5 miliarde de parametri și imagini de rezoluție 64x64 și un model de difuzie upsampling cu 1.5 miliarde de parametri pentru creșterea rezoluției la 256x256. Pe dataset-ul MS-COCO, soluția obține cel mai bun FID comparativ cu soluțiile AttnGan, DM-GAN, DF-GAN, LAFITE, DALL-E.

Modelul dă gresă înăsă la construirea imaginilor bazate pe scenarii imposibile și neobișnuite. Imaginile generate sunt, de cele mai multe ori, suprarealiste.

3.4 Alte soluții relevante

3.4.1 CLIP

Modelul CLIP [RKH⁺21] antrenează un text encoder și un image encoder pentru a putea prezice perechile corecte text-imagine din etapa de training. În faza de testare, text encoderul antrenat anterior sintetizează o clasificare lineară zero-shot prin encodarea descrierilor obiectelor din clasa target 3.12.

3.4.2 LAFITE : Generare de imagini folosind antrenare pe dataset-uri fără etichete

Acest model [ZZC⁺21] propune primul model pentru generate text-to-image al cărui proces de învățare se bazează doar pe imagini, nu și pe text. Metoda folosește spațiul semantic multi-modal al modelului pre-trained CLIP: nevoia text-conditionării este atenuată de generarea de text features din image features. Algoritmul generează perechi de psuedo-features în spațiul multi-modal.

Din punct de vedere al costurilor, un scenariu în care datele de training necesită doar imagini este ideal pentru că procesul de colectare a datelor devine mai ușor și mai rapid.

LAFITE funcționează eficient pe o gamă largă de configurații de generare text-to-image, cum sunt language-free, zero-shot și învățarea complet supervizată.

Generarea de Pseudo Text-Features

În proiectarea acestei soluții language-free se folosește modelul CLIP pentru obținerea pseudo-descrierii unei imagini date. Pentru că CLIP a fost antrenat pe un dataset de milioane de perechi text-imagine, asemănările între perechile imagine-text similare sunt maximezate.

În generarea de tip zero-shot, LAFITE depășește soluțiile DALL-E și CogView pe dataset-ul COCO. În generarea standard, complet supervizată, LAFITE depășește multe dintre metodele State-of-The-Art de până atunci.

Există și alte lucrări științifice (exemplu: [RDN⁺22], [SWR20], [Ble], [PNDA21]) care abordează problema generării imaginilor și care combină mai multe metode menționate anterior, însă niciuna nu prezintă rezultate relevante pentru această lucrare.

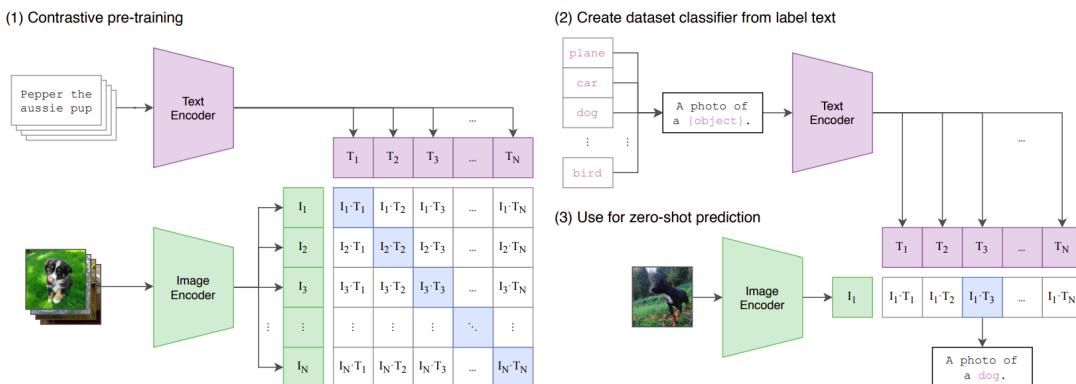


Figura 3.12: CLIP training and testing processes [RDN⁺22]

3.5 Direcții neexplorate

Analizând toate metodele relevante în rezolvarea problemei generării de imagini se observă că cele mai bune rezultate sunt obținute de rețelele de tip generativ. Se remarcă o îmbunătățire majoră de la utilizarea unui model generativ simplu,

la folosirea unei combinații de model generativ împreună cu un model de optimizare de tip Contrastive Language-Image Pre-Training (CLIP).

Cu toate acestea, niciuna din soluțiile analizate nu abordează problema procesării textului input. Toate soluțiile anterioare encodează în mod direct textul input și îl leagă la embedding-ul imaginii candidat. În următorul capitol se vor dezbatе soluția și abordarea propuse.

Capitolul 4

Soluția propusă

În urma analizării soluțiilor anterioare, am concluzionat că folosirea unui model de Generative Adversarial Network pentru generarea de design-uri de tatuaje este cea mai potrivită opțiune datorită performanței și tipului de imagini generate de aceste modele.

Datorită faptului că modelele sunt antrenate pe un număr finit de clase de obiecte și entități, un text input precum **divan** ar putea fi total neconcludent și necunoscut de către algoritmul nostru, pe când un text input precum **canapea** ar putea fi recunoscut de către acesta.

Se va folosi descrierea initială împreună cu alte câteva reformulări ale acesteia ca și input pentru mai multe instanțe ale algoritmului generativ. Reformulările vor fi obținute prin înlocuirea substantivelor și adjecțiivelor din descrierea initială cu sinonimele acestora.

Imaginiile generate de fiecare instanță a soluției vor fi "jurizate" cu ajutorul modelului CLIP pe baza unor cuvinte cheie care descriu, în mod general orice tatuaj (exemplu: artistic, colorful, stylized image, tattoo). Imaginea care va obține cel mai bun scor pe aceste cuvinte cheie, va fi aleasă ca imagine finală.

În mod adițional, se vor aplica 2 filtre pe această imagine finală, astfel încât utilizatorul să poată dispune de 3 variante din care poate alege: imagine finală originală, imaginea finală cu un filtru alb-negru, imaginea finală cu un filtru gri.

4.1 Specificații tehnice

Pentru obținerea descrierilor similare cu descrierea input, sunt înlocuite fiecare substantiv și adjecтив din textul input cu sinonime ale acestora. Acest lucru este posibil cu ajutorul librăriilor precum Spacy și nltk. Spre exemplu, pentru textul input "**a red carpet with a happy dog on it**" se obțin descrierile similare "**a burgundy carpet with a cheerful puppy on it**" și "**a red rug with a merry puppy on it**". Aceasta este componenta de extindere a spațiului lexical acoperit de descrierea input.

```
#-- initialDescription este descrierea introdusa de utilizator
#-- initialDescriptionWords conine cuvintele din descrierea initiala
```

```

initialDescriptionWords = re.split(',|_|_!_', initialDescription)

similarDescription1 = initialDescription
similarDescription2 = initialDescription

nlp = spacy.load("en_core_web_sm")
for word in initialDescription:
    if len(word) > 0:
        doc = nlp(word)
        if doc[0].pos_ == 'NOUN' or doc[0].pos_ == 'ADJ':
            syns = wordnet.synsets(word)
            #-- add synonyms
            similarDescription1 = similarDescription1.replace(word,
                syns[0].lemma_names()[0])
            similarDescription2 = similarDescription2.replace(word,
                syns[1].lemma_names()[0])

#-- se returneaza descrierea initiala impreuna cu alte 2 descrieri
#-- similare cu aceasta initialDescription, similarDescription1,
#-- similarDescription2

```

Fiecare dintre aceste descrieri și optional, o imagine de start, sunt folosite ca input pentru modelul generativ.

Pentru componenta rețelei generative se folosește modelul Vector Quantized Generative Adversarial Network, propus în lucrarea [ERO20]. Pentru a eficientiza din punct de vedere al timpului, se folosește un model pretrained pentru VQGAN. Există modele pretrained pe mai multe dataset-uri, însă soluția propusă în această lucrare folosește un model antrenat pe dataset-ul *IMAGENET 16384*. Acest dataset conține peste 14 milioane de imagini, adnotate manual și peste 20000 de categorii (conține categorii precum *căpsună*, *tablou*, *trandafir*).

Datorită faptului că un tatuaj nu este o clasă de entități, ci este un atribut al unei imagini (o imagine poate fi sau nu un design pentru un tatuaj; nu există o imagine reprezentativă pentru cum arată un tatuaj), modelul nostru trebuie să cunoască îndeajuns de multe clase astfel încât să poată satisface cerințele textului input al utilizatorului și să poată genera o imagine potrivită pentru a deveni un tatuaj.

Modelul VQGAN fie folosește imaginea de start primită ca parametru la apelarea metodei de generare, fie generează o imagine cu random noise. Plecând de la această imagine, modelul obține treptat mai multe imagini candidat. Fiecare din imaginile candidat este îmbunătățită cu ajutorul modelului CLIP astfel: imaginea este crop-uită, peste aceste părți din imagine se aplică mici modificări (noising, fliping, color jitter), se calculează media loss-ului și se modifică aceste părți din imagine în funcție de valoare medie a loss-ului. Loss-ul este diferența sferică pătratică dintre embedding-ul imaginii candidat și embedding-ul textului input; cu alte cuvinte, loss-ul calculat cu ajutorul CLIP ne spune cât de bine se potrivesc imaginea cu textul, comparând-se valorile embedding-urilor acestora din spațiul latent.

Acest proces se repetă pentru un număr stabilit de iterării, apoi se obține imaginea finală. Pentru că se rulează mai multe instanțe care generează câte o imagine, fiecare cu un text input diferit, se obțin mai multe imagini candidat.

Tot cu ajutorul modelului CLIP se testează similaritatea imaginilor cu diferite cuvinte cheie precum *colorful*, *tattoo*, *artistic*. Aceste cuvinte cheie sunt atribuite ale imaginilor generate care le fac sau nu să fie potrivite pentru design-uri de tatuaje.

CLIP ne returnează un scor pentru fiecare imagine, pentru fiecare cuvânt cheie. Imaginea care obține cel mai mare scor total (suma scorurilor pentru toate cuvintele cheie) este imaginea finală trimisă către utilizator. Aceasta este componenta de ranking a imaginilor candidat.

În mod adițional, se obțin alte două imagini din această imagine finală, o imagine alb negru și o imagine gri. Acestea sunt obținute pentru a ajuta utilizatorul și/sau artistul tatuator să vizualizeze blueprint-ul imaginii finale.

4.2 FlowChart-uri pentru soluția propusă

Diagrama 4.1 urmărește Flow-ul generării unui design de tatuaj de la textul input al utilizatorului, la imaginile finale returnate. În această diagramă sunt surprinse componenta de extindere a spațiului lexical al textului input și componenta de ranking a imaginilor candidat. Componenta de generare care constă în folosirea celor două modele VQGAN și CLIP este detaliată în diagrama 4.2.

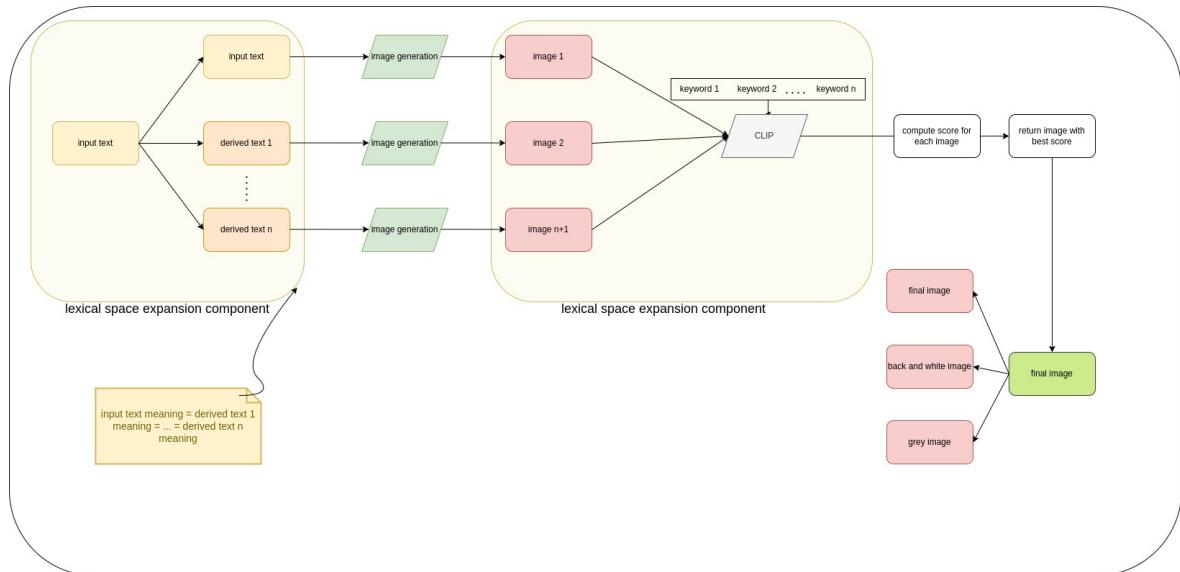


Figura 4.1: Tattoo Design Image Generation Flow

4.3 Implementarea soluției

Soluția este implementată în limbajul de programare Python datorită multitudinii și varietății de librării oferite în domeniul Machine Learning și Computer

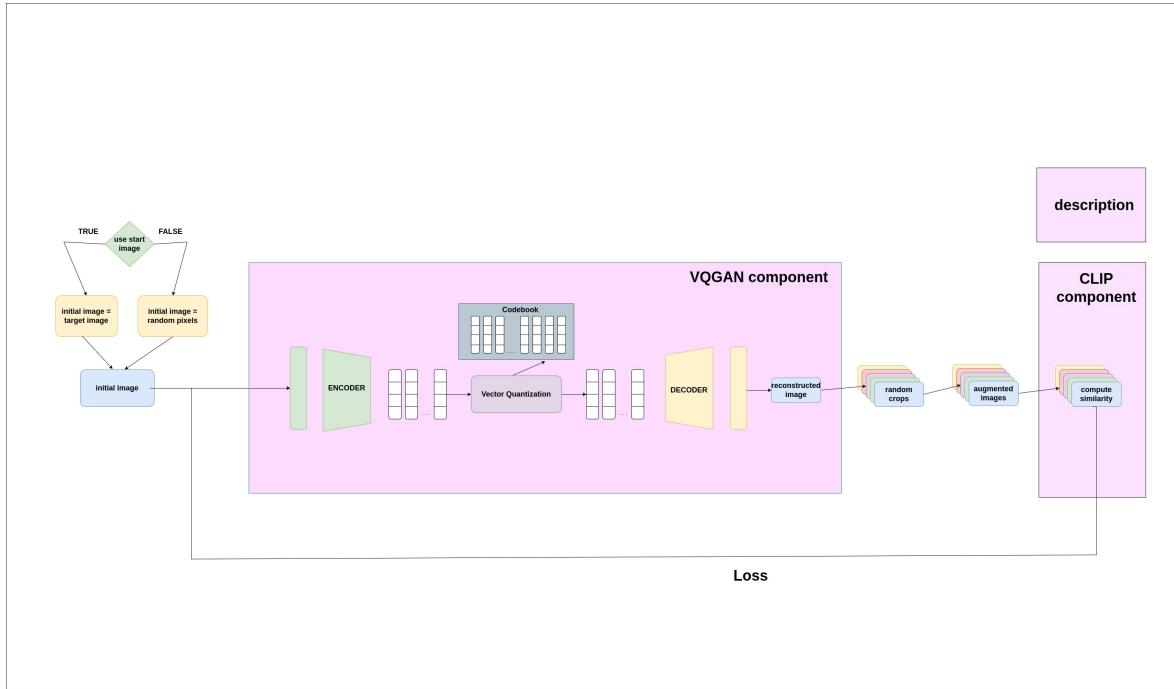


Figura 4.2: VQGAN image generation and CLIP optimization

Vision. Se folosește abordarea soluției în [CBK⁺22] pentru algoritmul generativ și prelucrarea imaginilor candidat.

Soluția este implementată într-un API care primește print-un POST call descrierea tatuajului dorit și, optional, o imagine de start, folosită ca referință în generarea design-ului final. Imaginea de start poate fi o imagine ce se dorește a fi modificata într-un design de tatuaj sau o imagine cu un tatuaj actual care se dorește a fi acoperit/transformat.

4.4 Specificarea și testarea API-ului

Generarea unui design de tatuaj este disponibila prin accesarea endpoint-ului:

[POST] **/tattoo-generation**

În corpul Request-ului este trimis un obiect JSON (JavaScript Object Notation) conținând perechi de tip cheie-valoare. În tabelul 4.1 sunt specificate cheile și tipurile de date necesare corpului Request-ului. Un exemplu de obiect JSON trimis endpoint-ului de generare tattoo design este ?? .

Listing 4.1: Exemplu de JSON folosit la Generarea unui tatuaj

```

1  {
2      "description": "a red carpet | Picasso",
3      "startImagePath":
4          ".../..../..../opt/lampp/htdocs/tattootopyImages/steps/0200.png",
5      "uniqueTattooId": 2
6  }
  
```

Name	Data type	Required/Optional	Description
description	String	required	descrierea tatuajului dorit
startImagePath	String	optional	locația imaginii de start
uniqueTattooId	Long	required	id-ul tatuajului folosit la salvarea imaginii finale

Tabela 4.1: The Request Body / Corpul Request-ului

Exemplu de Request:

```
$ curl --location --request POST
  'http://127.0.0.1:4567/tattoo-generation' \
--header 'Content-Type: application/json' \
--data-raw '{
  "description": "a red carpet | Picasso",
  "startImagePath":
    "../../../../opt/lampp/htdocs/tattootopyImages/steps/0200.png",
  "uniqueTattooId": 2
}'
```

Exemplu de Response:

Endpoint-ul răspunde cu un HTTPStatus și cu un JSON care conține path-urile către imaginile generate. Tabelul 4.2 cuprinde informații referitoare la tipurile de răspunsuri și semnificațiile acestora, iar tabelul 4.3 conține specificarea JSON-ului returnat de endpoint.

Status	Description
OK / 200	generarea s-a efectuat cu succes
BAD_REQUEST / 400	date primite sunt invalide
INTERNAL_SERVER_ERROR / 500	eroare a serverului

Tabela 4.2: The Response Types / Tipurile de răspunsuri

Locațiile imaginilor trimise către endpoint pot fi atât path-uri absolute, în cazul în care imaginea se află pe aceeași mașină cu serverul, cât și locații URL. Locațiile returnate în JSON sunt exclusiv URL, astfel încât clientul să poată vizualiza imaginile de pe orice device.

Name	Data type	Required/Optional	Description
finalImagePath	String	required	locatia imaginii originale finale
finalImagePathBW	String	required	locatia imaginii alb-negru generate
finalImagePathG	String	required	locatia imaginii gri generate

Tabela 4.3: The Response Body / Corpul răspunsului

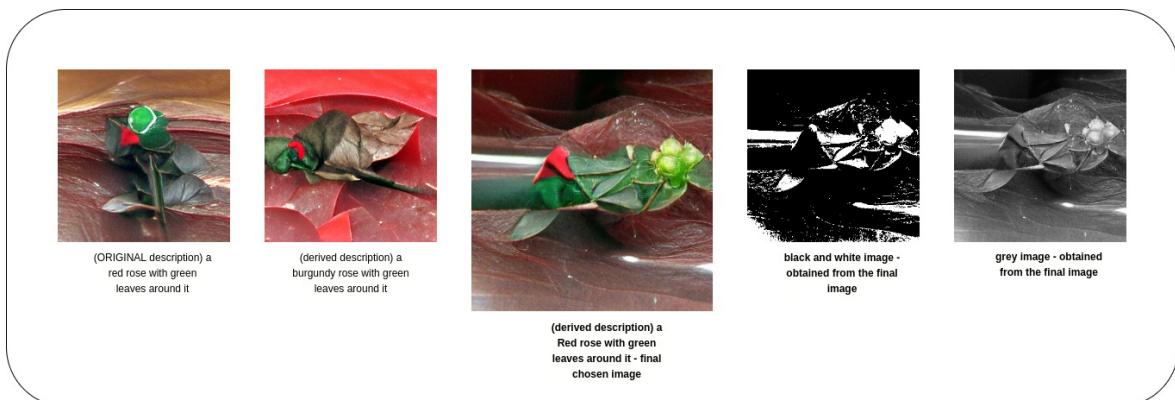


Figura 4.3: Generare condiționată de text - 500 iteratii per imagine - 300x300 px

4.5 Rezultatele generării

Mai jos sunt prezentate rezultatele câtorva dintre generările la care a fost supusă soluția prezentată, alături de descrierile și imaginile de start (unde este cazul) folosite pentru generarea acestora.

Pentru generările din imaginile 4.3 și 4.4 s-au folosit 3 instanțe de generare, adică din textul input s-au obținut 3 imagini (una cu descrierea originală și două cu descrieri derivate din aceasta). Pentru fiecare generare se specifică în descrierea acesteia numărul de iteratii în urma căror a fost obținută imaginea finală. De la stânga la dreapta, în colajul 4.3 se află cele trei imagini generate, în centru, imaginea finală, aleasă datorită scorului obținut, și cele două imagini cu filtre alb-negru și, respectiv, gri.

Următoarele imagini au fost generate în urma a 200 de iteratii. În partea stângă se află imaginea finală.

Descrierea poate conține stilul sau numele artistului în stilul căruia se dorește să fie generată imaginea (exemplu: *a flower / watercolor, a flower by Andy Warhol*).

4.6 Evaluarea performantei API-ului propus

Pentru evaluarea performantei au fost comparate rezultatele obținute de modelul nostru de dezvoltare lexicală cu rezultatele modelelor State-of-The-Art in

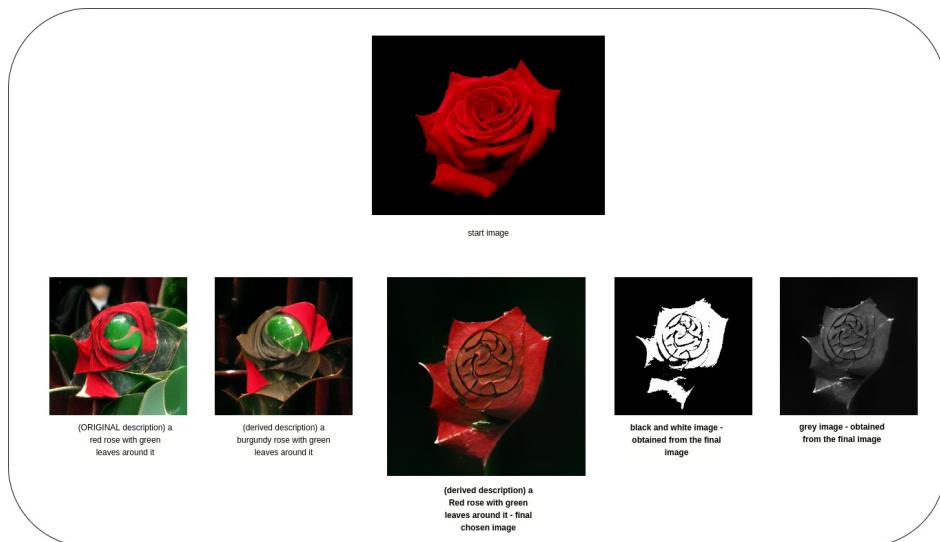


Figura 4.4: Generare condiționată de text și imagine - 500 iteratii per imagine - 600x600 px



Figura 4.5: Generare condiționată de text - 200 iteratii per imagine - 600x600 px

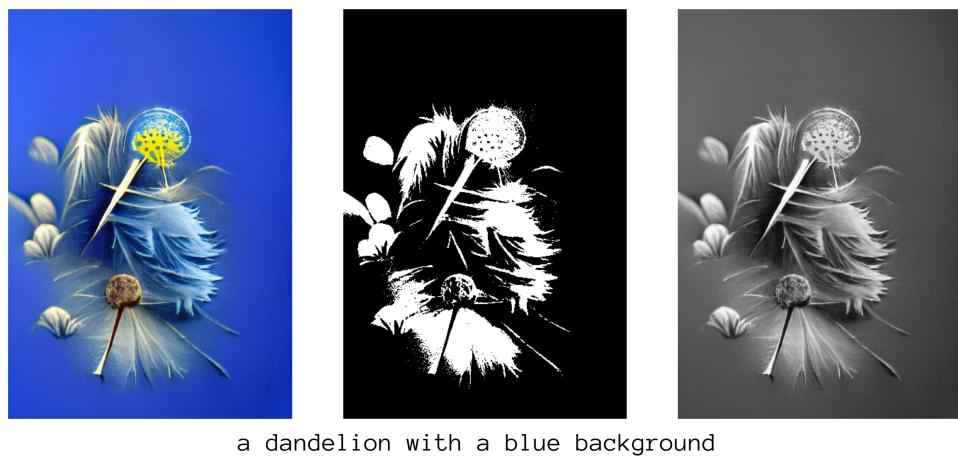


Figura 4.6: Generare condiționată de text - 200 iteratii per imagine - 400x600 px

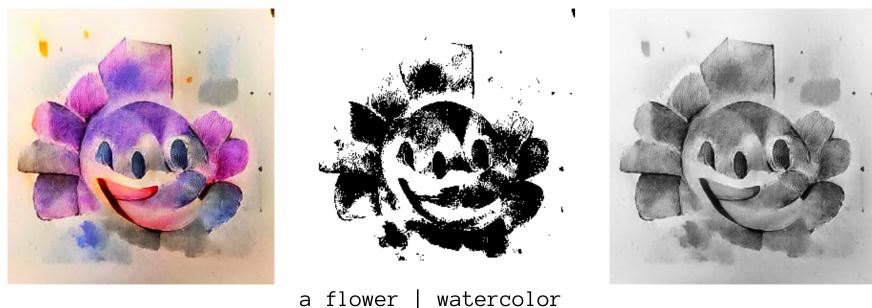


Figura 4.7: Generare condiționată de text - 200 iteratii per imagine - 300x300 px

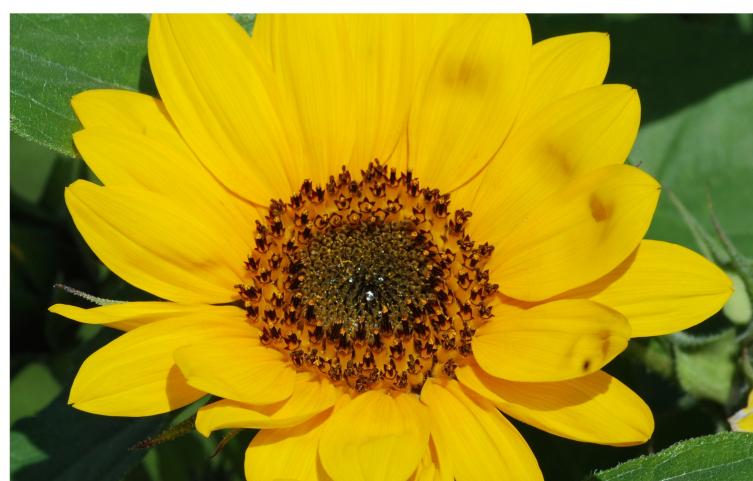
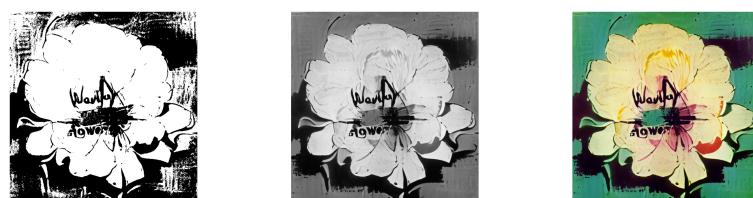


Figura 4.8: Generare condiționată de text și imagine - 300 iteratii per imagine - 300x300 px

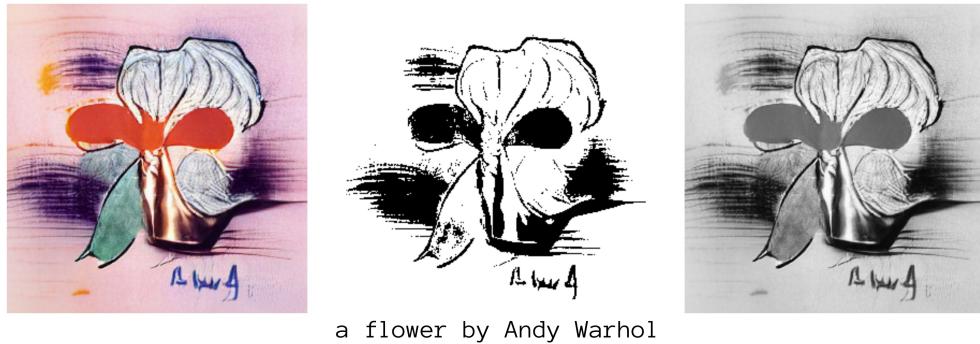


Figura 4.9: Generare condiționată de text - 600 iteratii per imagine - 300x300 px

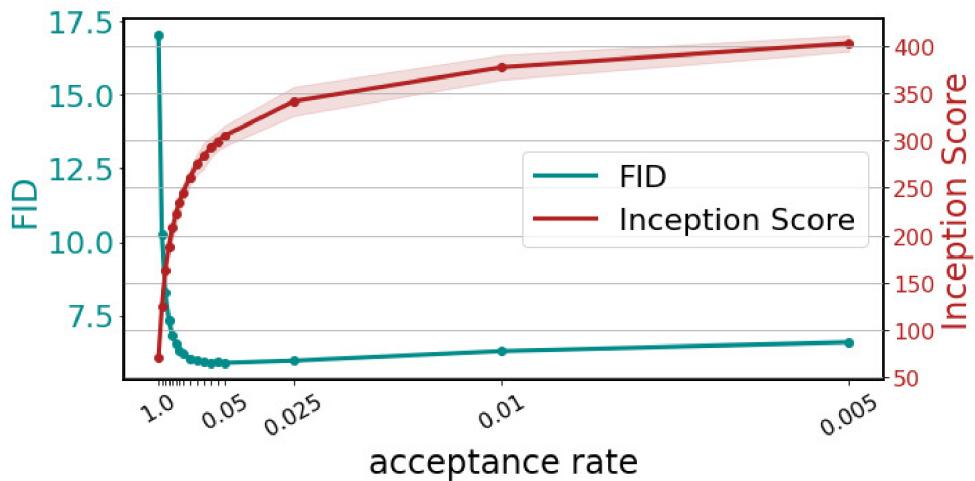


Figura 4.10: Scorurile obținute la antrenarea modelului de VQGAN - preluate din [ERO20]

generarea de imagini. Se menționează faptul ca niciunul din modelele anterioare nu este orientat spre generarea de imagini de tip tatuaj. Se observă că adăugarea componentei de dezvoltare lexicală și a componentei de ranking a imaginilor candidat pentru alegerea imaginii finale duce la rezultate mai bune și mai aproape de forma unui design de tatuaj decât generarea clasica de imagini pe unele cazuri (4.11). În 4.6 se află imaginile produse de soluțiile State-of-The-Art în generarea de imagini și imaginile produse de soluția noastră orientată spre imagini de tip design de tatuaj. De la stânga la dreapta imaginile sunt produse de soluțiile minDALL-E [SKB21], GLIDE [NDR⁺21], VQGAN+CLIP [CBK⁺22] și soluția noastră.

Datorită faptului că am folosit un model pretrained, nu a fost posibilă efectuarea calculelor metricilor Inception Score (IS) [BS18] și Fréchet Inception Distance (FID) [YZD21]. Scorurile obținute pentru antrenarea modelului de VQGAN folosit în această lucrare se află în lucrarea [ERO20].



Figura 4.11: the universal library trending on artstation



Figura 4.12: a charcoal drawing of a cathedral

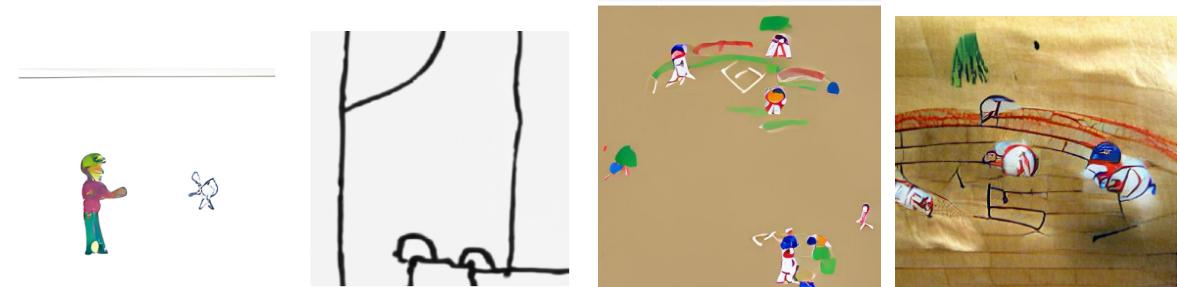


Figura 4.13: a child's drawing of a baseball game

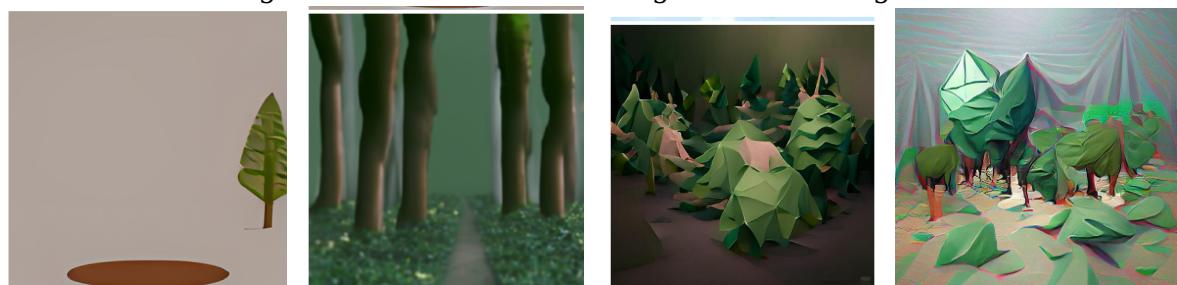


Figura 4.14: a forest rendered in low poly

Capitolul 5

Aplicație

Pentru a facilita accesarea soluției propuse de către utilizatori, propun apelarea API-ului specificat anterior printr-o aplicație care să permită utilizatorilor să genereze și să gestioneze design-uri de tatuaje. Astfel, se testează scalabilitatea și comportamentul algoritmului pentru un număr mare de clienți.

Acest capitol cuprinde descrierea comportamentului, arhitecturii și componentelor aplicației propuse, alături de explicații privind fiecare caz de utilizare disponibil utilizatorilor. Ultima secțiune surprinde modul de rezolvare al unei probleme des întâlnite în aplicație prin intermediul unui şablon de programare.

5.1 Aplicația propusă

Propun o aplicație de tip WEB, prin care un utilizator își poate genera unul sau mai multe tatuaje pe baza unei descrieri text și/sau a unei imagini, folosind API-ul propus anterior. Aplicația este compusă din două părți, o parte de API constituind backend-ul, și o parte Web constituind front-end-ul.

Aplicația este menită să ofere o interfață user-friendly și să permită vizualizarea și administrarea tatuajelor generate anterior. Platforma permite vizualizarea imaginilor în forma originală (nu și a imaginilor cu filtrele *black and white* și *grey* aplicate).

5.2 Arhitectura aplicației

Pentru dezvoltarea aplicației se folosește o arhitectură stratificată datorită simplității și scalabilității puse la dispoziție de aceasta. În acest capitol, vor fi prezentate cazuile de utilizare însotite de diagrame de secvență care relevă interacțiunile componentelor arhitecturale ale aplicației.

5.2.1 Comportament - cazuri de utilizare

Platforma permite crearea și accesarea unui cont. În acest cont sunt disponibile următoarele funcționalități:

- generarea condiționată de text și/sau imagine a unui design de tatuaj
- vizualizarea listei de generări anterioare
- schimbarea nivelului de acces al unei generări de tatuaj (public/privat)
- stergerea unei generări de tatuaj
- vizualizarea imaginilor generate în scop demonstrativ (demo images)
- logout

Diagrama 5.1 ilustrează aceste cazuri de utilizare.

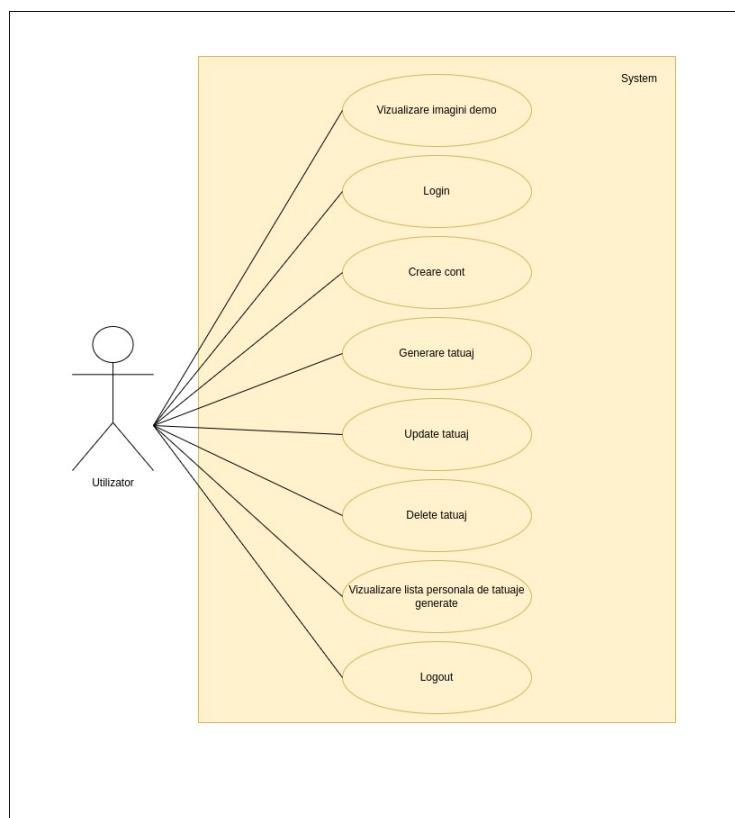


Figura 5.1: Diagrama cazurilor de utilizare ale aplicației propuse

5.2.2 Structura - diagrama de componente

Se folosește o arhitectură REST (REpresentational State Transfer) pentru construirea unui serviciu Web. Design-ul de client-server permite dezvoltarea independentă a componentelor front-end și backend și îmbunătățește portabilitatea și accesibilitatea platformei.

Request-urile sunt trimise de pe partea de client pe partea de server și sunt interceptate de către TattooController. Aceasta definește metode care gestionează toate tipurile de Request-uri necesare și trimit rezultate însotite de statusuri înapoi către client. Datorită naturii Stateless a arhitecturii REST, requesturile conțin totă informația de care are nevoie serverul pentru a putea răspunde și computa rezultatul.

Comunicarea dintre client și server se face prin Call-uri API și cu ajutorul entităților de transfer a datelor (Data Transfer Objects - DTOs).

Serverul este construit pe o arhitectură stratificată, având componentele și rolurile următoare:

- The domain layer: conține entitățile aplicatiei
- The infrastructure layer: conține clasele responsabile pentru managementul entităților, repository-urile, legătura cu baza de date, aici sunt definite și implementate operațiile CRUD pentru entitățile din domeniu
- The application layer: conține service-urile care se ocupă de logica aplicatiei
- The presentation layer: conține Controller-ele responsabile de a intercepta și răspunde la Request-urile clientilor

Într-o astfel de arhitectură stratificată, componentele dintr-un anumit strat au acces doar la componentele din straturile inferioare, adică dependențele merg într-o singură direcție, de la stratul de presentation la stratul domain. Arhitectura stratificată încurajează simplitatea, consistența și aplicarea principiilor OOP (separarea principiilor, principul open-close) și ușurează testarea independentă a straturilor.

Figura 5.2 exemplifică componentele arhitecturale ale aplicației.

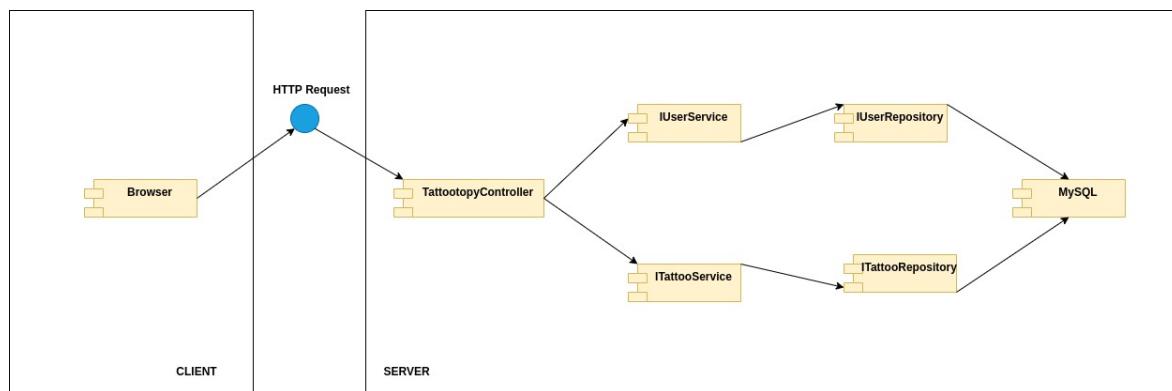


Figura 5.2: Diagrama de componente

Acest tip de arhitectură se pretează foarte bine pe aplicațiile care nu suferă modificări ale entităților din domeniu, cum este și cazul acestei aplicații.

5.2.3 Interacțiuni ale componentelor în cazurile de utilizare

Caz de utilizare - Vizualizare imagini Demo

Vizualizarea imaginilor Demo 5.3 este o funcționalitate accesibilă utilizatorilor fără autentificare. Lista de tatuaje generate în scop demonstrativ este afișată pe pagina de Overview. La crearea componentei de Overview, se trimit un Request către server prin care se cere lista tuturor imaginilor Demo. Controller-ul apelează ITattooService pentru a primi o lista cu entități de tip Demo Tattoo pe care le transformă în entități de tip Data Transfer Object. Lista de Data Transfer Objects este trimisă înapoi către client împreună cu statusul OK (200).

Service-ul comunică cu ITattooRepository pentru a primi lista tuturor tatuajelor din baza de date; apoi Service-ul selectează doar tatuajele de tip demo și le trimit către Controller.

Se trimit înapoi la client răspunsuri astfel:

- Status OK (200) și Lista de entități Tattoo: happy flow
- INTERNAL_SERVER_ERROR (500): se întâmpina o excepție

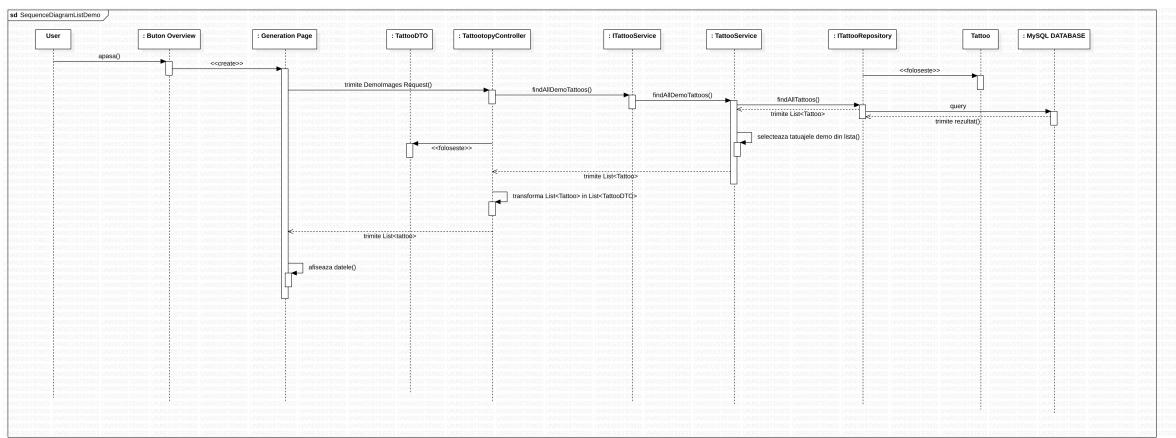


Figura 5.3: Caz de utilizare - Vizualizarea imaginilor generate în scop demonstrativ

Caz de utilizare - Login

Diagrama 5.4 exemplifica flow-ul funcționalității de Login.

Utilizatorul accesează pagina de autentificare prin click pe butonul de Login. Aici, completează formularul cu datele necesare (email și parola) și apoi apasă butonul de Login. În urma apăsării butonului, câmpurile completate sunt verificate pentru a nu fivide. În cazul în care câmpurile complete sunt invalide, utilizatorul este anunțat printr-un mesaj de warning că datele introduse nu sunt corespunzătoare. În cazul în care acestea sunt valide, se va trimite un Request de Login către server cu credențialele furnizate de user (email și parola hash-uită pentru securitatea datelor).

Controller-ul primește acest Request și face o validare suplimentară a email-ului. În cazul în care acesta este valid, se caută în baza de date utilizatorul cu această adresă de email și se verifică dacă parola hash-uită primită de la client se potrivesc cu cea stocată în baza de date.

Se trimit înapoi la client statusuri astfel:

- BAD_REQUEST (400): email-ul primit de la client este null
- NOT_FOUND (404): nu există user cu adresa de email primită
- UNAUTHORIZED (401): parolele nu se potrivesc
- OK (200): autentificarea a reușit
- INTERNAL_SERVER_ERROR (500): se întâmpina o excepție

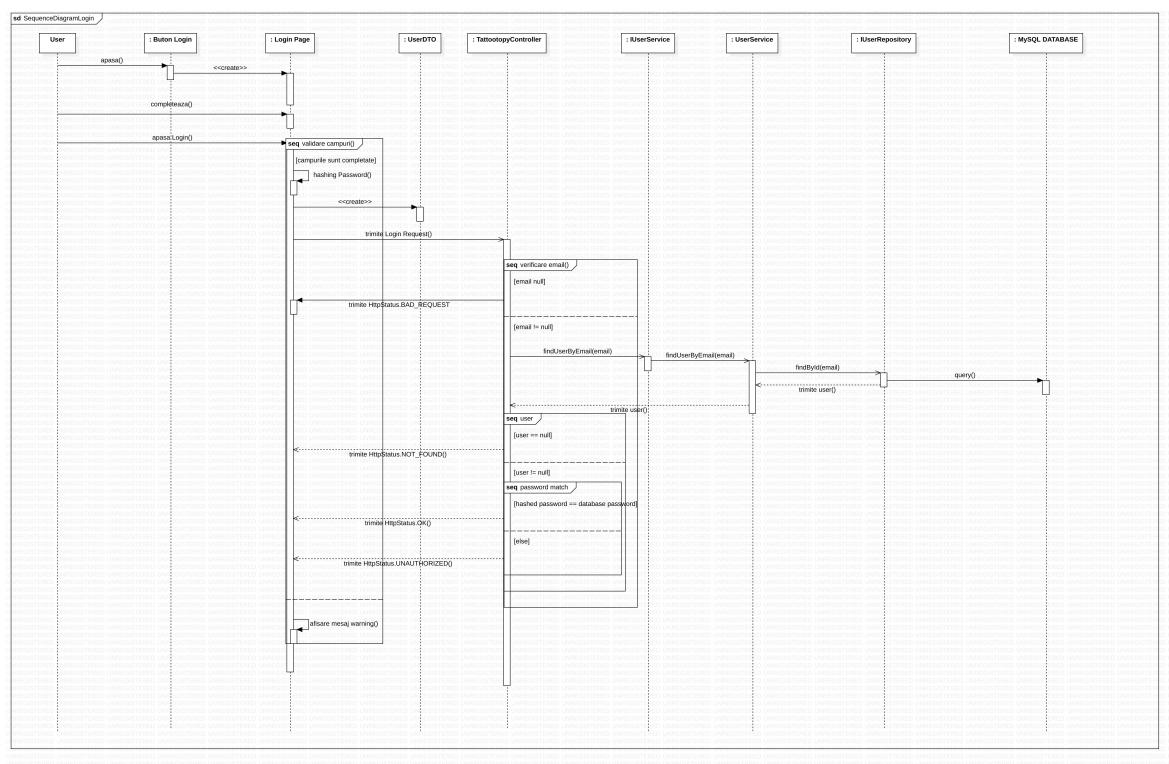


Figura 5.4: Caz de utilizare Login

Caz de utilizare - Creare cont

Utilizatorii noi își pot crea cont prin click pe butonul de Register din pagina principală. Utilizatorul va fi redirectionat către pagina de Creare cont unde este necesară completarea câmpurilor nume, prenume, adresa de email și parolă.

La apăsarea butonului Register, câmpurile sunt verificate pentru a nu fi goale, iar în caz afirmativ, se trimit un Request de creare cont către server. Request-ul conține aceste date furnizate de client (parola fiind hash-uită).

Pe partea de server se face o validare adițională a datelor. În cazul în care datele sunt invalide, se trimit un status de BAD_REQUEST (400). Altfel, se caută

În baza de date un utilizator cu adresa de email primită. Dacă există un utilizator cu adresa de email primită, se returnează un status de UNAUTHORIZED (401) deoarece nu pot exista doi utilizatori cu aceeași adresa de email (adresa de email este cheie primară în baza de date pentru tabelul utilizatorilor). Dacă adresa de email nu este folosită de niciun alt utilizator, se creează contul și se returnează statusul OK (200). În cazul întâmpinării oricărora exceptii, se returnează INTERNAL_SERVER_ERROR (500).

Diagrama de secvență 5.5 ilustrează interacțiunile dintre componente pentru acest caz de utilizare.

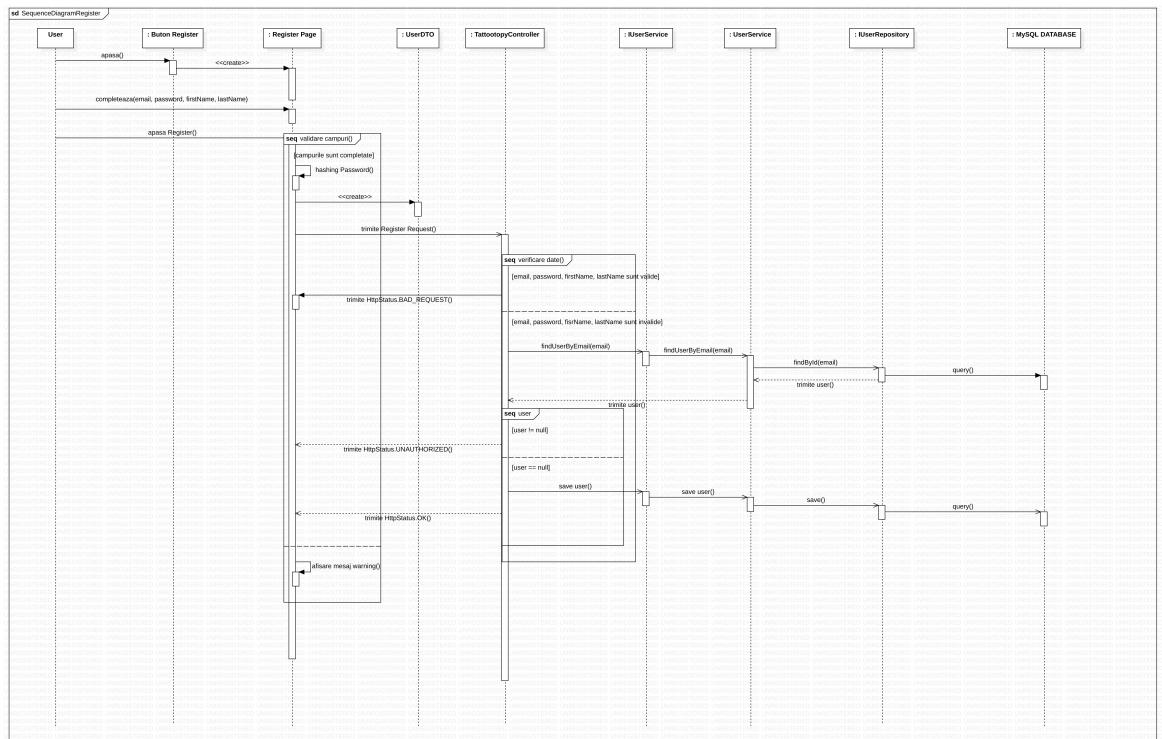


Figura 5.5: Caz de utilizare - Creare Cont (Register)

Caz de utilizare - Generare tatuaj

Funcționalitatea principală a aplicării este generarea de tatuaje. După autentificare, utilizatorul este redirectionat către pagina *Home*, unde își poate genera un tatuaj. Astfel, userul va completa câmpurile unei form cu datele necesare (descrierea tatuajului și/sau o imagine de start) și va apăsa butonul GO. În urma apăsării butonului GO, se va trimite un Request de tipul Generare Tatuaj (există două tipuri de Request-uri făcute către backend: generare fără imagine de start, generare cu imagine de start) conținând informațiile din câmpurile completate de utilizator și adresa de email a acestuia.

TattooopyController, Controller-ul serverului, va căuta User-ul cu adresa de email primită, va crea un nou Tattoo cu specificațiile cerute de client, va salva acest Tattoo în baza de date și va atribui acest tatuaj utilizatorului corespunzător

(va crea în baza de date legătura dintre utilizator și tatuaj). Aceste operații sunt efectuate de ITattooService și ITattooRepository.

ITattooService se ocupă de generarea imaginii și comunică cu serverul Python. Către serverul de Python sunt trimise descrierea și imaginea de start (optional). Se primesc înapoi path-urile către imaginile finale generate (imaginea finală originală, imaginea finală alb-negru, imaginea finală gri). După ce se primește rezultatul de la serverul Python, Service-ul TattooService apelează la TattooRepository pentru a updata în baza de date entitatea Tattoo cu path-ul către imaginea finală (imaginea originală).

Deoarece procesul generării unei imagini poate dura între câteva minute și câteva zeci de minute în funcție de tipul de execuție (pe Central Processing Unit sau pe Graphics Processing Unit) și dimensiunea imaginii, utilizatorul este anunțat printr-un email de finalizarea generării tatuajului său. La finalizarea generării tatuajului, ITattooService apelează la clasa MailSenderHelper pentru a trimite un email userului cu imaginile generate. Imaginea originală este accesibilă și în lista de generări personale ale utilizatorului.

Pentru comportamente din afara Happy Flow-ului pe partea de server a aplicației, Controller-ul returnează statusuri astfel:

- BAD_REQUEST (400): nu se găsește niciun utilizator cu adresa de email furnizată în corpul Request-ului sau adresa de email furnizată este vidă
- INTERNAL_SERVER_ERROR (500): se întâmpina o excepție

Întregul proces și interacțiunile componentelor sunt ilustrate în Diagrama 5.6

Caz de utilizare - Vizualizare listă de generări personale

Utilizatorul autentificat va putea vizualiza o lista cu toate imaginile generate anterior (imaginile finale originale). La accesarea paginii **Home** de către utilizator, se va face un Request către server prin care se cere lista cu entități de tip Tattoo ale utilizatorului logat. Controller-ul apelează la IUserService pentru a primi aceasta lista de entități.

IUserService apelează la metoda findById() din IUserRepository, având ca parametru al metodei adresa de email a userului. Datorita mapărilor făcute de ORM, se va obține entitatea User care are ca și instance variable o lista de tatuaje. Aceasta lista de entități Tattoo este transformată într-o listă de entități TattooDTO și este trimisă către clientul Web în corpul răspunsului.

Caz de utilizare - Update nivel de confidențialitate

Utilizatorul are posibilitatea să schimbe nivelul de confidențialitate a unei generări 5.8. La vizualizarea listei de generări, în pagina **Home**, utilizatorul va putea vizualiza cîte un buton toggle pentru fiecare din generările sale. Acest buton schimbă nivelul de confidențialitate al generării respective de la private (default value) la public și invers.

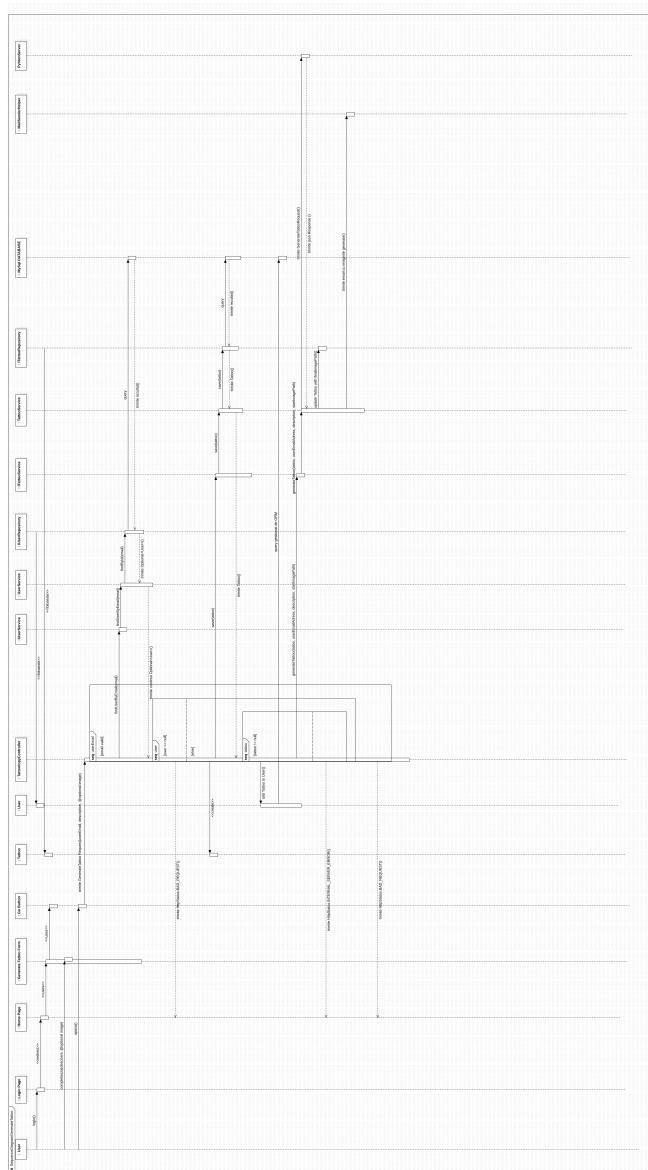


Figura 5.6: Caz de utilizare - Generare tatuaj

La actionarea butonului, se trimite un Request către server. Request-ul conține informații referitoare la id-ul item-ului din lista (Tattoo id) și la starea butonului toggle.

Controller-ul din server verifică validitatea datelor primite și apelează funcția de update din ITattooService. Implementarea Service-ului, TattooService, va căuta entitatea Tattoo în baza de date prin intermediul Repository-ului. În cazul în care nu a fost găsit niciun Tattoo cu id-ul primit, metoda din Service returnează false (caz exceptional - nu ar trebui să apară). Altfel, se updatează entitatea și se returnează true.

Controller-ul interpretează valoarea booleană primită de la Service și trimite un răspuns clientului astfel:

- BAD_REQUEST (400): id-ul primit este invalid
- NOT_FOUND (404): nu există Tattoo cu id-ul primit

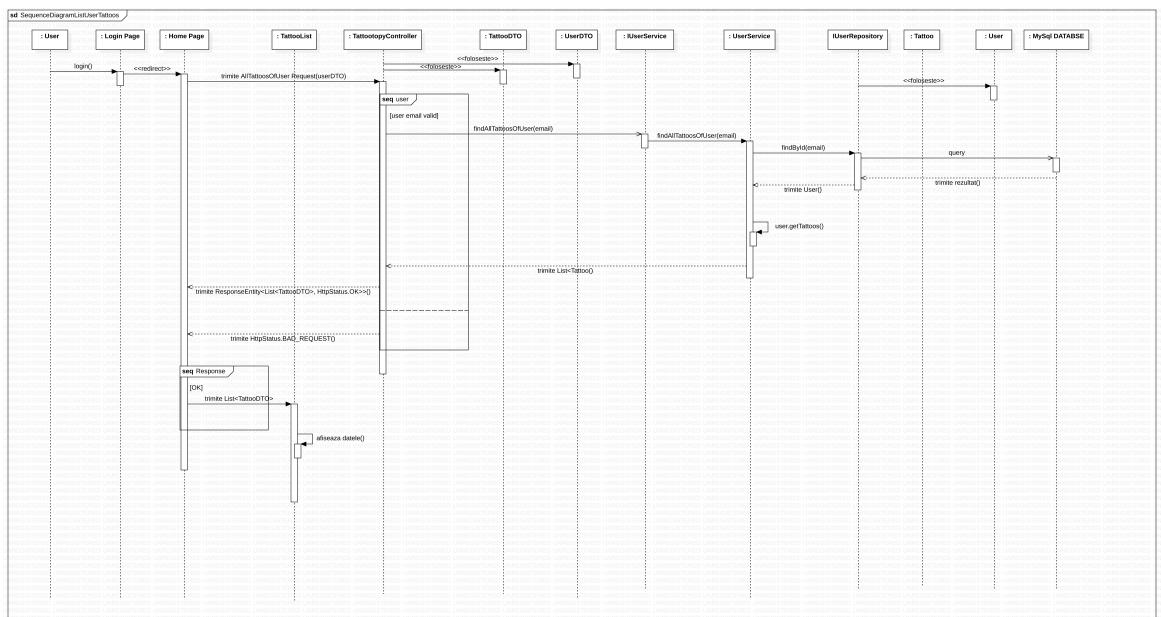


Figura 5.7: Caz de utilizare - Vizualizare lista generări anterioare

- OK (200): update-ul a reușit
- INTERNAL_SERVER_ERROR (500): se întâmpina o excepție

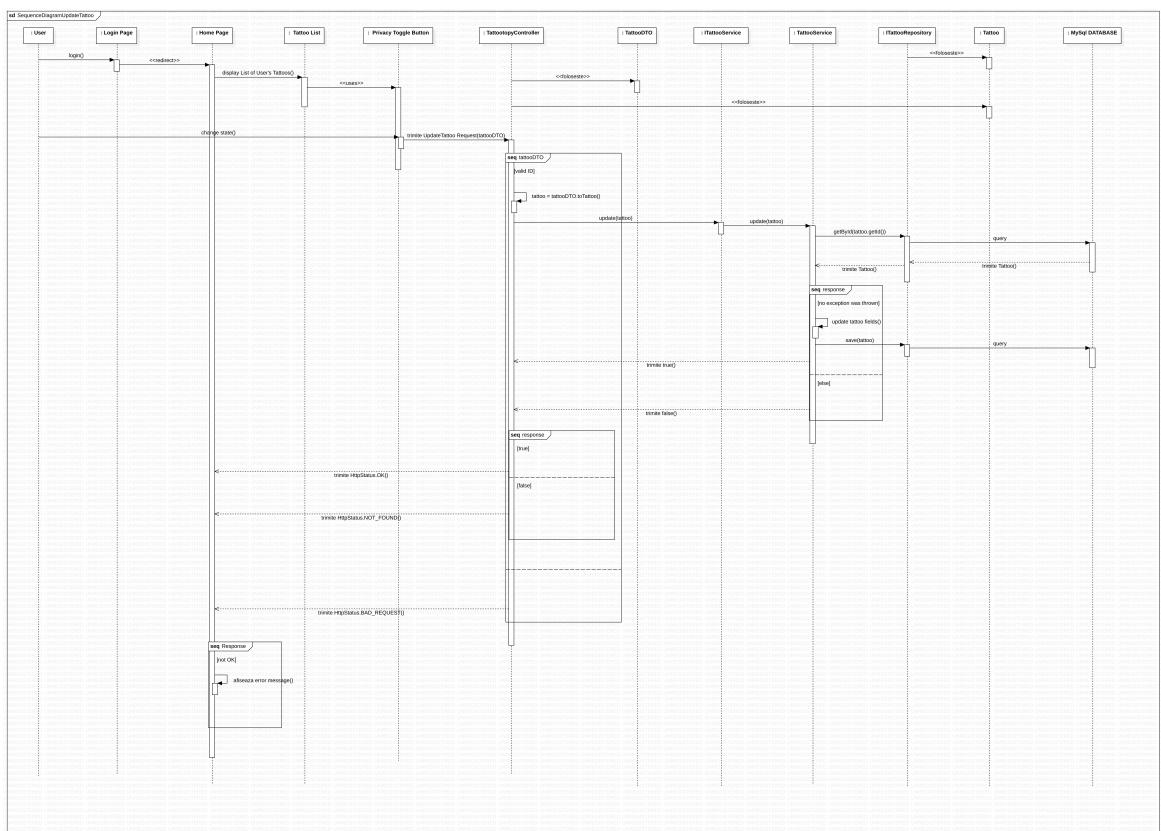


Figura 5.8: Caz de utilizare - Update nivel de confidentialitate a generării

Caz de utilizare - Delete generare tatuaj

Utilizatorul poate să steargă o generare personală 5.9. La vizualizarea listei de generări, utilizatorul va putea actiona un button de delete. La apăsarea acestui buton, se trimite un Request către server conținând id-ul entității.

Controller-ul serverului efectuează funcționalitatea de delete și răspunde clientului web cu unul din statusurile:

- NOT_FOUND (404): nu există Tattoo cu id-ul primit
- OK (200): delete-ul a reușit
- INTERNAL_SERVER_ERROR (500): se întâmpină o excepție

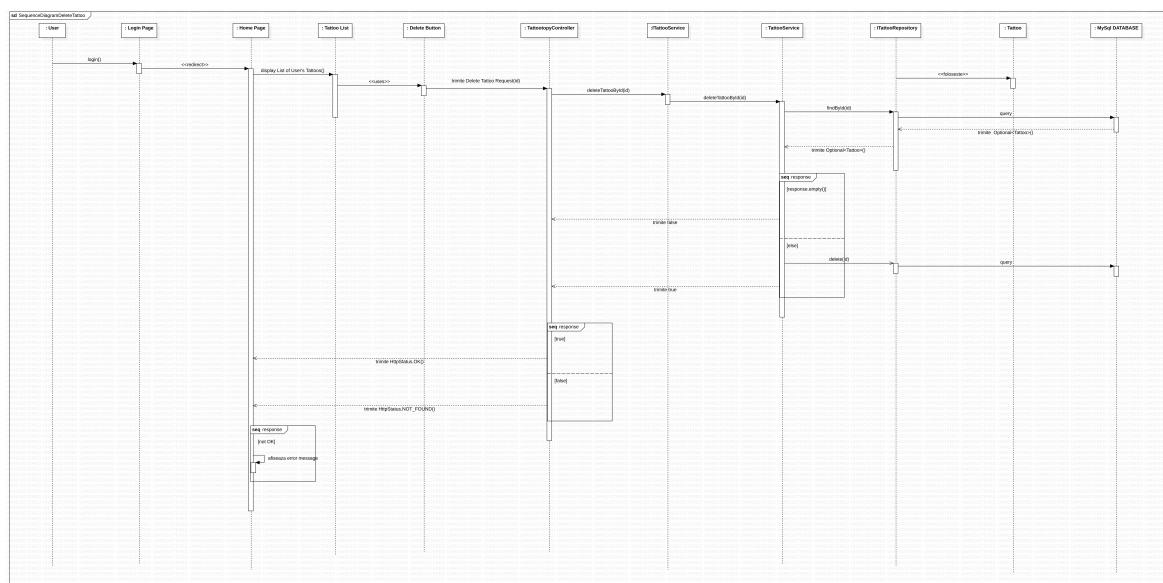


Figura 5.9: Caz de utilizare - Delete generare

5.3 Design Patterns

Se folosește **Pattern-ul Singleton** la initializarea clasei MailSenderHelper. Mai mult, se folosește lazy initialization, ceea ce face ca obiectul să fie creat doar atunci când este nevoie de el și nu atunci când sunt încărcate clasele în memorie de către JVM (ca în cazul initializării eager). De fiecare dată când este nevoie de o instanță a clasei MailSenderHelper, Singleton Pattern ne ajută să refolosim aceeași instanță și astfel se previne risipa de resurse.

Mai mult, pentru că trimitera mail-urilor se face asincron, se folosește modelul de 'Lazy initialization with Double check locking'. Acest mecanism rezolvă problema sincronizării clasice a metodei care poate încetini performanța codului deoarece metoda nu poate fi accesată de mai multe thread-uri simultan.

Prin folosirea metodei de 'Lazy initialization with Double check locking', threadurile vor aștepta doar dacă acceseză simultan metoda de getMailSenderHelperInstance() când instanța de clasa este nulă.

```
public class MailSenderHelper {  
  
    private static MailSenderHelper mailSenderHelper = null;  
  
    \\ aici se initializeaza alte instance variables  
  
    private MailSenderHelper() {  
    }  
  
    public static MailSenderHelper getMailSenderHelperInstance() {  
        if (mailSenderHelper == null) {  
            synchronized (MailSenderHelper.class) {  
                if (mailSenderHelper == null) {  
                    // if the instance is null, then we initialize it  
                    mailSenderHelper = new MailSenderHelper();  
                }  
            }  
        }  
        return mailSenderHelper;  
    }  
  
    synchronized public void sendmail(String emailAddress, String path){  
        // aici se trimite emailul  
    }  
}
```

Capitolul 6

Implementarea aplicației

Pentru implementarea aplicației au fost alese Java și JavaScript ca și limbaje de programare pentru serverul aplicației, respectiv pentru clientul web. Ca și formă de persistentă a fost aleasă o bază de date relatională datorită potrivirii cu framework-urile folosite pe partea de server a aplicatiei.

În următoarele secțiuni sunt înglobate toate detaliile tehnice referitoare la modul de implementare al aplicației, modul de comunicare al componentelor, specificațiile endpoint-urilor, limbajele de programare, framework-urile și librăriile folosite. Ultima secțiune conține imagini cu paginile web ale aplicației și manualul de utilizare al platformei web.

6.1 Implementarea arhitecturii abstracte

Serverul aplicației este implementat în Java. Acesta conține entitățile User și Tattoo. Clasa User are atributele email, hashedPassword (parola hash-uită), firstName și lastName; iar clasa Tattoo are atributele id (identificatorul unic), description (descrierea imaginii), path (locația imaginii), isPublic (indică dacă imaginea poate fi accesată de către toți utilizatorii) și isDemo (indică dacă imaginea a fost generată în scop demonstrativ).

Pentru ambele entități s-au implementat clase de Repository și Interfețe pentru Repository, clase de Service și Interfețe pentru Service. Clasa TattooController se ocupă de toate call-urile de API necesare funcționării aplicației. Diagrama de clase a aplicației este ilustrată în imaginea 6.1.

6.2 Limbajele de programare și tehnologiile folosite

Următoarele secțiuni prezintă librăriile și framework-urile folosite în dezvoltarea aplicației, incluzând motivația alegerii lor și beneficiile aduse de acestea.

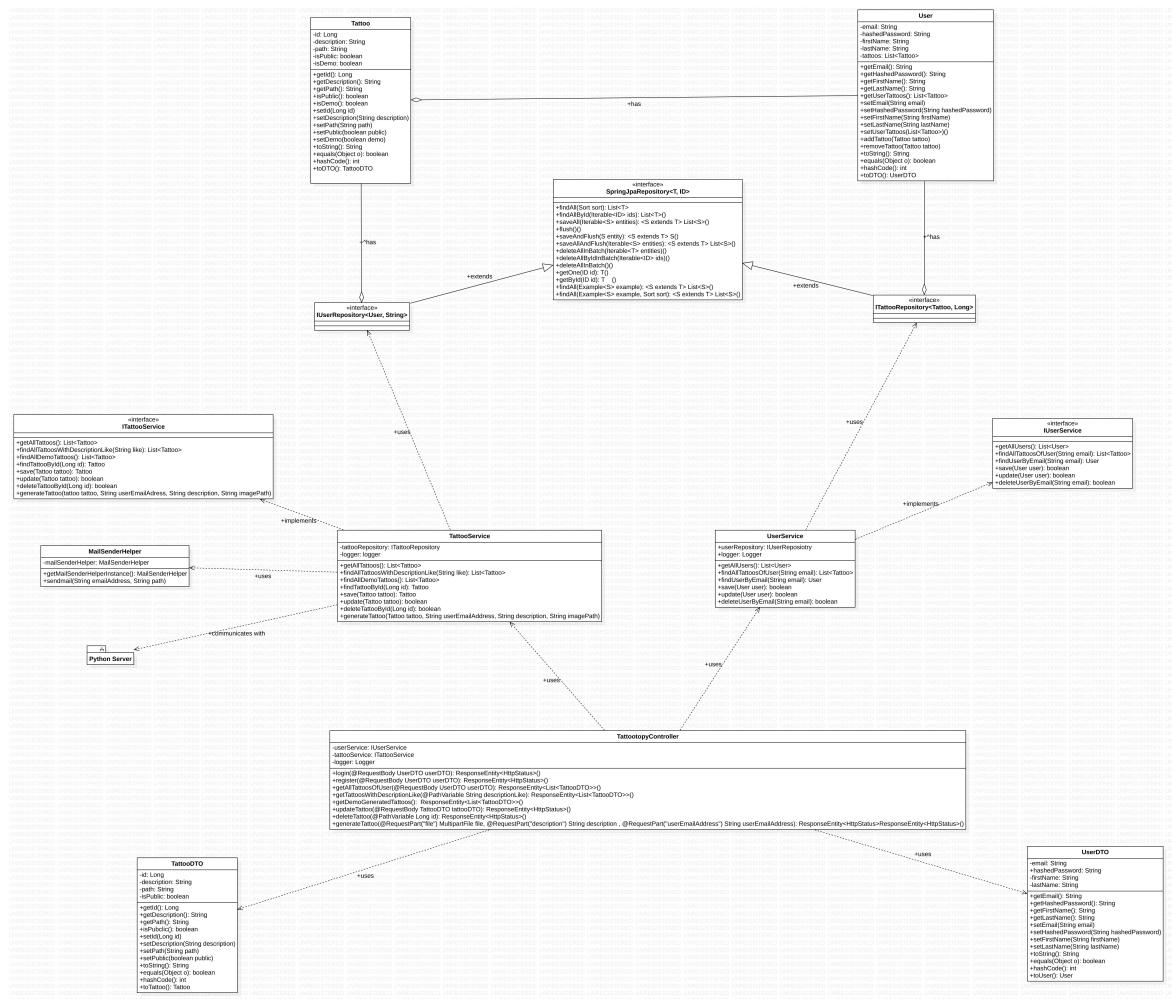


Figura 6.1: Diagrama de clase - server JAVA

6.2.1 Serverul aplicării

Pentru partea de server a aplicației au fost considerate mai multe limbaje de programare, cum ar fi C#, Java, Golang și Python. În cele din urmă Java a fost ales ca limbaj de programare datorită popularității sale, fiind cel mai folosit limbaj prioritar de programare de către web back-end developers [Jet].

Un alt motiv pentru care Java este o alegere atrăgătoare este datorita Spring Framework. Spring Boot este cel mai folosit framework când vine vorba de crearea de aplicații web [Jet].

De asemenea Spring Boot vine cu avantajul de a crea foarte ușor un proiect, folosind Spring Initializr. In pachetul de început sunt adăugate și tool-uri utile, cum ar fi Hibernate (un ORM bine documentat și ușor de folosit pentru baze de date precum MySql), Spring JPA Repositories (repository-uri generate automat ce scurtează semnificativ timpul de dezvoltare a proiectului), și tool-urile de implementare a call-urilor de API (ce ușurează crearea și extinderea unui server).

Baza de date aleasa este MySql, aplicația fiind una web care, potențial, va stoca multe date ale utilizatorilor (inclusiv adrese de e-mail, imagini generate etc.). MySql este eficient și ușor scalabil când vine vorba de aplicații cu o multi-

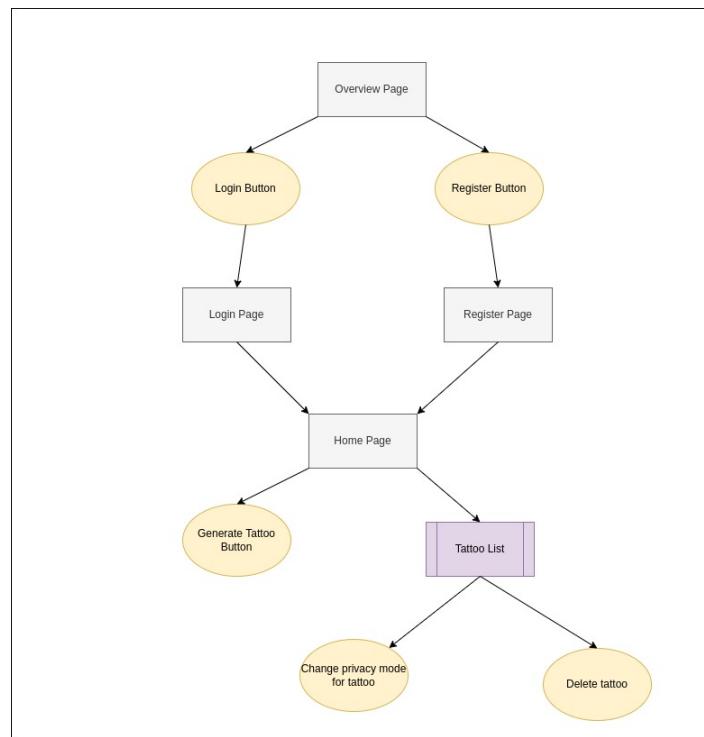


Figura 6.2: Diagrama de componente - client React JS

tudine de clienti.

6.2.2 Serverul de generare de imagini

Pentru serverul responsabil cu generarea de tatuaj a fost folosit Python, datorita librăriilor și a tool-urilor ce vin cu acest limbaj de programare. Folosind Python avem acces la tool-uri și librarii precum:

- CUDA (pentru paralelizarea algoritmului de ML pe placa video și pe procesor)
- Flask (pentru crearea și configurarea call-urilor de API)
- spaCy și nltk (pentru modificarea textului, folosind librarii de sinonime și detectare de substantive/adjective)
- Pytorch, CLIP (pentru dezvoltarea algoritmului de machine learning)

6.2.3 Clientul Web

Pentru partea de client web am ales sa folosesc JavaScript, ReactJS. React este cea mai populară librărie folosită pentru crearea de pagini web, construită și documentată de Facebook Open Source.

Pentru componente de user interface a fost folosit Material UI, o librărie de componente folosită pentru dezvoltarea aplicațiilor precum Netflix, Spotify, Nasa etc. Pentru controlarea call-urilor API se folosește librăria axios.

6.2.4 Specificații endpoint-uri

Login

Login-ul este folosit pentru autentificarea unui utilizator. Este necesară existența unui cont asociat acestui utilizator pentru ca autentificarea să fie făcută cu succes.

Endpoint-ul pentru funcționalitatea de Login este:

[POST] **/login**

În corpul Request-ului este trimis un obiect JSON (JavaScript Object Notation) conținând perechi de tip cheie-valoare. În tabelul 6.1 sunt specificate cheile JSON-ului și tipurile de date necesare corespunzătoare acestor chei. Un exemplu de obiect JSON trimis endpoint-ului de Login este 6.1 .

Name	Data type	Required/Optional	Description
email	String	required	adresa de email a utilizatorului pentru care se face autentificarea
hashedPassword	String	required	parola hash-uită a utilizatorului pentru care se face autentificarea

Tabela 6.1: The Request Body / Corpul Request-ului

Listing 6.1: Exemplu JSON folosit la Login

```

1 {
2   "email": "janedoe@gmail.com",
3   "hashedPassword": "aaa",
4 }
```

Exemplu de Request:

```
$ curl --location --request POST
  'http://localhost:8080/tatt00topy/login' \
--header 'Content-Type: application/json' \
--data-raw '{
  "email": "janedoe@gmail.com",
  "hashedPassword": "aaa"
}'
```

Exemplu de Response:

Endpoint-ul răspunde cu un *ResponseEntity<HttpStatus>* în funcție de rezultatul autentificării. Tabelul 6.2 cuprinde informații referitoare la tipurile de răspunsuri și semnificațiile acestora.

Status	Description
OK / 200	autentificarea a reușit
BAD_REQUEST / 400	email-ul primit este null
UNAUTHORIZED / 401	parola nu este corectă
NOT_FOUND / 404	email-ul primit nu este asociat unui cont
INTERNAL_SERVER_ERROR / 500	eroare a serverului

Tabela 6.2: The Response Types / Tipurile de răspunsuri

Creare cont / Register

Register-ul este folosit pentru crearea unui cont nou pentru un utilizator. Adresa de email furnizată trebuie să nu fie folosită de un alt utilizator pentru ca funcționalitatea să se efectueze cu succes.

Endpoint-ul pentru funcționalitatea de Register este:

[POST] **/register**

În corpul Request-ului este trimis un obiect JSON (JavaScript Object Notation) conținând perechi de tip cheie-valoare. În tabelul 6.3 sunt specificate cheile JSON-ului și tipurile de date necesare fiecărei chei. Un exemplu de obiect JSON trimis endpoint-ului de Register este 6.2 .

Name	Data type	Required/Optional	Description
email	String	required	adresa de email folosită pentru crearea contului
hashedPassword	String	required	parola hash-uită asociată contului
firstName	String	required	prenumele utilizatorului
lastName	String	required	numele utilizatorului

Tabela 6.3: The Request Body / Corpul Request-ului

Listing 6.2: Exemplu JSON folosit la Register

```

1 {
2   "email": "janedoe@gmail.com",
3   "hashedPassword": "bbb",
4   "firstName": "Jane",
5   "lastName": "Doe"
6 }
```

Exemplu de Request:

```
$ curl --location --request POST
  'http://localhost:8080/tattoo/register' \
--header 'Content-Type: application/json' \
--data-raw '{
  "email": "janedoe@gmail.com",
  "hashedPassword": "bbb",
  "firstName": "Jane",
  "lastName": "Doe"
}'
```

Exemplu de Response:

Endpoint-ul răspunde cu un *ResponseEntity<HttpStatus>* în funcție de rezultatul creării contului. Tabelul 6.4 cuprinde informații referitoare la tipurile de răspunsuri și semnificațiile acestora.

Status	Description
OK / 200	contul a fost creat
BAD_REQUEST / 400	request-ul contine câmpuri invalide/vide
UNAUTHORIZED / 401	adresa de email este deja folosită
INTERNAL_SERVER_ERROR / 500	eroare a serverului

Tabela 6.4: The Response Types / Tipurile de răspunsuri

Vizualizare imagini demonstrative

Funcționalitatea de vizualizare a imaginilor generate cu scop demonstrativ este disponibilă la endpoint-ul:

[GET] **/demo**

Call-ul nu necesită transmiterea datelor adiționale în corpul Request-ului.

Exemplu de Request:

```
$ curl --location --request GET 'http://localhost:8080/tattoo/demo'
```

Exemplu de Response:

Endpoint-ul răspunde cu un *ResponseEntity<List< TattooDTO >>* ce conține lista de entități *TattooDTO* generate în scop demonstrativ (*is_Demo = true*). Tabelul 6.5 cuprinde informații referitoare la tipurile de răspunsuri și semnificațiile acestora.

Status	Body	Description
OK / 200	ArrayList<TattooDTO>	succes
INTERNAL_SERVER_ERROR / 500	null	eroare a serverului

Tabela 6.5: The Response Types / Tipurile de răspunsuri

Vizualizarea tatuajelor generate de un utilizator

Vizualizarea imaginilor generate de un anumit utilizator este disponibilă prin accesarea endpoint-ului:

[POST] **/all-tattoos-of-user**

În corpul Request-ului este trimis un obiect JSON conținând perechi de tip cheie-valoare. În tabelul 6.6 sunt specificate cheile și tipurile de date necesare corpului Request-ului. Un exemplu de obiect JSON trimis endpoint-ului de Register este 6.3 .

Name	Data type	Required/Optional	Description
email	String	required	adresa de email a utilizatorului

Tabela 6.6: The Request Body / Corpul Request-ului

Listing 6.3: Exemplu JSON folosit în corpul Request-ului

```

1 {
2   "email": "janedoe@gmail.com"
3 }
```

Exemplu de Request:

```
$ curl --location --request POST
  'http://localhost:8080/tattoo/api/all-tattoos-of-user' \
--header 'Content-Type: application/json' \
--data-raw '{
  "email": "janedoe@gmail.com"
}'
```

Exemplu de Response:

Endpoint-ul răspunde cu un *ResponseEntity<List<TattooDTO>>* și conține lista de entități TattooDTO. Tabelul 6.7 cuprinde informații referitoare la tipurile de răspunsuri și semnificațiile acestora.

Listing 6.4: Exemplu JSON returnat

Status	Body	Description
OK / 200	ArrayList<TattooDTO>	succes
BAD_REQUEST / 400	null	adresa de email este invalidă
INTERNAL_SERVER_ERROR / 500	null	eroare a serverului

Tabela 6.7: The Response Types / Tipurile de răspunsuri

```

1 [ 
2   {
3     "id": 101,
4     "description": "a cat",
5     "path": "https://i.ibb.co/ZN2rQgc/4.jpg",
6     "public": true
7   },
8   {
9     "id": 102,
10    "description": "a colorful piece of art",
11    "path": "https://i.ibb.co/ZN2rQgc/5.jpg",
12    "public": true
13  }
14 ]

```

Update - Schimbare nivel de acces/privacy al unei generări

Schimbarea și updatarea nivelului de acces al unui tatuaj se face accesând endpoint-ul:

[POST] **/update-tattoo**

În corpul Request-ului este trimis un obiect JSON (JavaScript Object Notation) conținând perechi de tip cheie-valoare. În tabelul 6.8 sunt specificate cheile și tipurile de date necesare corpului Request-ului. Un exemplu de obiect JSON trimis endpoint-ului de Update este 6.5 .

Name	Data type	Required/Optional	Description
id	Long	required	id-ul tatuajului
isPublic	boolean	required	noul status al tatuajului

Tabela 6.8: The Request Body / Corpul Request-ului

Listing 6.5: Exemplu de JSON folosit la Update

```

1 {
2   "id": 72,
3   "isPublic": "TRUE"
4 }
```

Exemplu de Request:

```
$ curl --location --request POST
  'http://localhost:8080/tattoo/update-tattoo' \
--header 'Content-Type: application/json' \
--data-raw '{
  "id": 72,
  "isPublic": "TRUE"
}'
```

Exemplu de Response:

Endpoint-ul răspunde cu un *ResponseEntity<HttpStatus>* în funcție de rezultatul creării contului. Tabelul 6.9 cuprinde informații referitoare la tipurile de răspunsuri și semnificațiile acestora.

Status	Description
OK / 200	entitatea a fost updatată
BAD_REQUEST / 400	id-ul primit este invalid/nul
NOT_FOUND / 404	nu există entitate Tattoo cu id-ul primit
INTERNAL_SERVER_ERROR / 500	eroare a serverului

Tabela 6.9: The Response Types / Tipurile de răspunsuri

Delete - Stergerea unei generări

Stergerea unui tatuaj generat anterior se face accesând endpoint-ul:

[DELETE] **/delete-tattoo/{id}**

Request-ul necesită un PathVaribale de tip întreg care reprezinta id-ul entității Tattoo care va fi șters din baza de date.

Exemplu de Request:

```
$ curl --location --request DELETE
  'http://localhost:8080/tattoo/delete-tattoo' \
--header 'Content-Type: application/json' \
--data-raw '56'
```

Exemplu de Response:

Endpoint-ul răspunde cu un *ResponseEntity<HttpStatus>* în funcție de rezultatul operației de ștergere. Tabelul 6.10 cuprinde informații referitoare la tipurile de răspunsuri și semnificațiile acestora.

Status	Description
OK / 200	entitatea a fost ștearsă cu succes
NOT_FOUND / 404	nu există entitate Tattoo cu id-ul primit
INTERNAL_SERVER_ERROR / 500	eroare a serverului

Tabela 6.10: The Response Types / Tipurile de răspunsuri

Generare tatuaj fără imagine de start

Funcționalitatea de generare tatuaj fără imagine de start este disponibilă prin accesarea endpoint-ului:

[POST] **/generate-tattoo-without-image**

În corpul Request-ului este trimis un obiect DataForm. În tabelul 6.11 sunt specificate partile componente și tipurile de date necesare Request-ului.

Request Part Name	Data type	Required/Optional	Description
description	String	required	descrierea tatuajului
userEmailAddress	String	required	adresa de email a utilizatorului

Tabela 6.11: The Request Body / Corpul Request-ului

Exemplu de Request:

```
$ curl --location --request POST
  'http://localhost:8080/tattootropy/generate-tattoo-without-image' \
--form 'description="a nice place"' \
--form 'userEmailAddress="janedoe@gmail.com"'
```

Exemplu de Response:

Endpoint-ul răspunde cu un *ResponseEntity<HttpStatus>* în funcție de rezultatul generării imaginii. Tabelul 6.12 cuprinde informații referitoare la tipurile de răspunsuri și semnificațiile acestora.

Status	Description
OK / 200	generarea a început
BAD_REQUEST / 400	request-ul conține date invalide
INTERNAL_SERVER_ERROR / 500	eroare a serverului

Tabela 6.12: The Response Types / Tipurile de răspunsuri

Generare tatuaj folosind o imagine de start

Funcționalitatea de generare tatuaj folosind o imagine de start este disponibilă prin accesarea endpoint-ului:

[POST] **/generate-tattoo-with-image**

În corpul Request-ului este trimis un obiect DataForm. În tabelul 6.13 sunt specificate partile componente și tipurile de date necesare Request-ului.

Request Part Name	Data type	Required/Optional	Description
file	MultipartFile	optional	imagină de start folosită pentru generarea tatuajului
description	String	required	descrierea tatuajului
userEmailAddress	String	required	adresa de email a utilizatorului

Tabela 6.13: The Request Body / Corpul Request-ului

Exemplu de Request:

```
$ curl --location --request POST
  'http://localhost:8080/tattootopy/generate-tattoo-with-image' \
--form 'file="//localhost/opt/lampp/htdocs/tattootopyImages/a.png"' \
--form 'description="a nice place"' \
--form 'userEmailAddress="janedoe@gmail.com"'
```

Exemplu de Response:

Endpoint-ul răspunde cu un *ResponseEntity<HttpStatus>* în funcție de rezultatul generării imaginii. Tabelul 6.14 cuprinde informații referitoare la tipurile de răspunsuri și semnificațiile acestora.

6.2.5 Testarea aplicatiei

Pentru testarea aplicatiei s-au aplicat metodele Unit Testing și Integration Testing. Au fost testate componentele: TattooController, TattooService, UserService, ITattooRepository, IUserRepository și au fost folosite framework-urile

Status	Description
OK / 200	generarea a început
BAD_REQUEST / 400	request-ul conține date invalide
INTERNAL_SERVER_ERROR / 500	eroare a serverului

Tabela 6.14: The Response Types / Tipurile de răspunsuri

de testare Junit, Mockito și DataJpaTest. Endpoint-urile au fost testate pentru acoperirea tuturor tipurilor de răspunsuri (ex: 200, 404, 500, etc.). De asemenea, acestea au fost testate și cu tool-ul Postman.

Testele au fost scrise astfel încât să acopere toate cazurile de utilizare, atât pe partea de Service, cât și pe partea de Repository.

6.3 Detalii de implementare

Cu toate că platforma nu permite vizualizarea imaginilor obținute din imaginea finală (imaginea cu filtru alb-negru și imaginea cu filtru gri), acesta sunt trimise utilizatorului printr-un email. Email-ul este trimis către utilizator, imediat ce se primește rezultatul de la serverul Python. Acesta conține un text standard și are atașate cele trei imagini finale (originala, alb-negru și gri) 6.15. Pentru trimitera email-urilor s-a creat o adresa de gmail sub numele aplicației *tattootropy@gmail.com*.

Pentru securitatea datelor, parola este hash-uită atât la autentificare, cât și la crearea contului de utilizator. De asemenea, pentru securitatea paginilor web, accesarea de către utilizatori neautentificați a URL-urilor care necesită autentificare duce la redirectionarea către pagina de **Overview**.

6.4 Persistența datelor

Pentru persistarea datelor se folosește o bază de date relațională de tip SQL (Structured Query Language). Am ales stocarea datelor într-o bază de date relațională datorită ușurinței de a crea relații între entități și de a popula și interroga tabelele.

MySQL este cel mai sigur sistem de management al bazei de date, fiind folosit în diverse aplicații web cunoscute precum: Facebook, Twitter și WordPress. Aceasta:

- oferă performanță și viteză în execuția query-urilor și a tranzacțiilor
- suportă toate comenziile de bază SQL
- este portabilă pentru că poate rula pe diferite platforme precum Linux, Solaris și Windows

- oferă siguranță securității și integrității datelor
- oferă beneficiile proprietăților ACID (Atomicity, Consistency, Isolation, Durability)

Astfel, am ales să folosim o bază de date MySQL pentru manipularea datelor aplicației. Baza de date a fost aleasă astfel încât să comunice ușor cu ORM-ul și framework-urile folosite pe partea de Java server. ORM-ul Hibernate oferă suport pentru MySQL și simplifică interacțiunea cu baza de date, oferind suport și documentație în ceea ce privește configurarea și customizarea entităților și a relațiilor dintre acestea.

Tabelele din baza de date sunt: users, tattoos și user_tattoos. Tipurile de date ale coloanelor tabelelor păstrează tipurile de date ale atributelor claselor de baza din diagrama de clase.

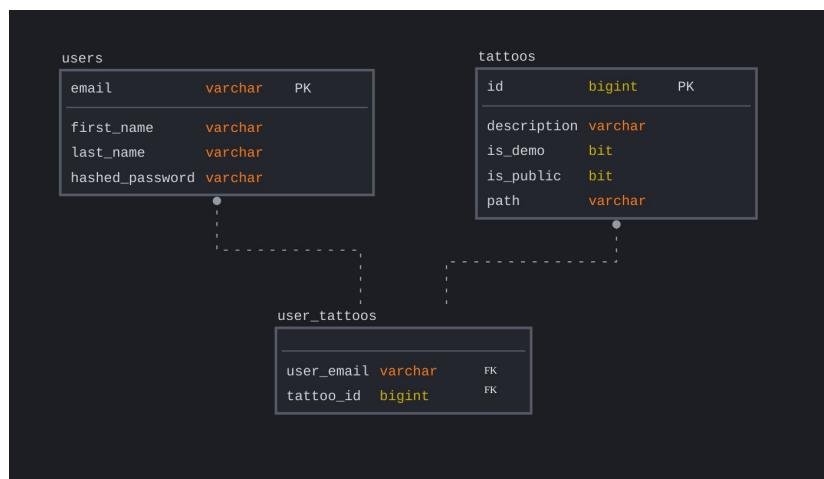


Figura 6.3: Diagrama bazei de date

Diagrama bazei de date 6.3 ilustrează tabelele și relațiile dintre acestea. Un user poate avea mai multe tatuaje, dar un tatuaj poate apartine unui singur user, ceea ce creează o relație de one-to-many între tabelele users și tattoos. Există tabelul de legătura users_tattoos care stocă această relație. Tabelul users_tattoos are două coloane:

- user_email: cheie străină aparținând tabelului users
- tattoo_id: cheie străină aparținând tabelului tattoos

Cheile străine din tabelul de legătura sunt chei primare în tabelele de origine. În tabelul **tattoos** sunt salvate atât datele despre tatuajele generate de utilizatori, cât și datele despre tatuajele generate în scop demonstrativ (acestea având flag-ul is_demo setat la 1 - TRUE). În tabelul **users** se salvează parola hash-uită în coloana hashed_password și nu parola originală din motive de securitate.

6.5 UX si manual de utilizare

La accesarea platformei web se va deschide pagina de **Overview** 6.4. Pagina poate fi accesata fară autentificare. Aici este disponibilă o listă cu imagini generate în scop demonstrativ alături de descrierile care le-au generat. În pagina de **Overview** apare în stânga-sus logo-ul aplicației 6.5.

La apăsarea butonului de **get inked** utilizatorul este redirectionat către pagina de **Login**. Butonul de **Login** deschide pagina de autentificare, iar butonul de **Register** deschide pagina pentru crearea unui cont nou.

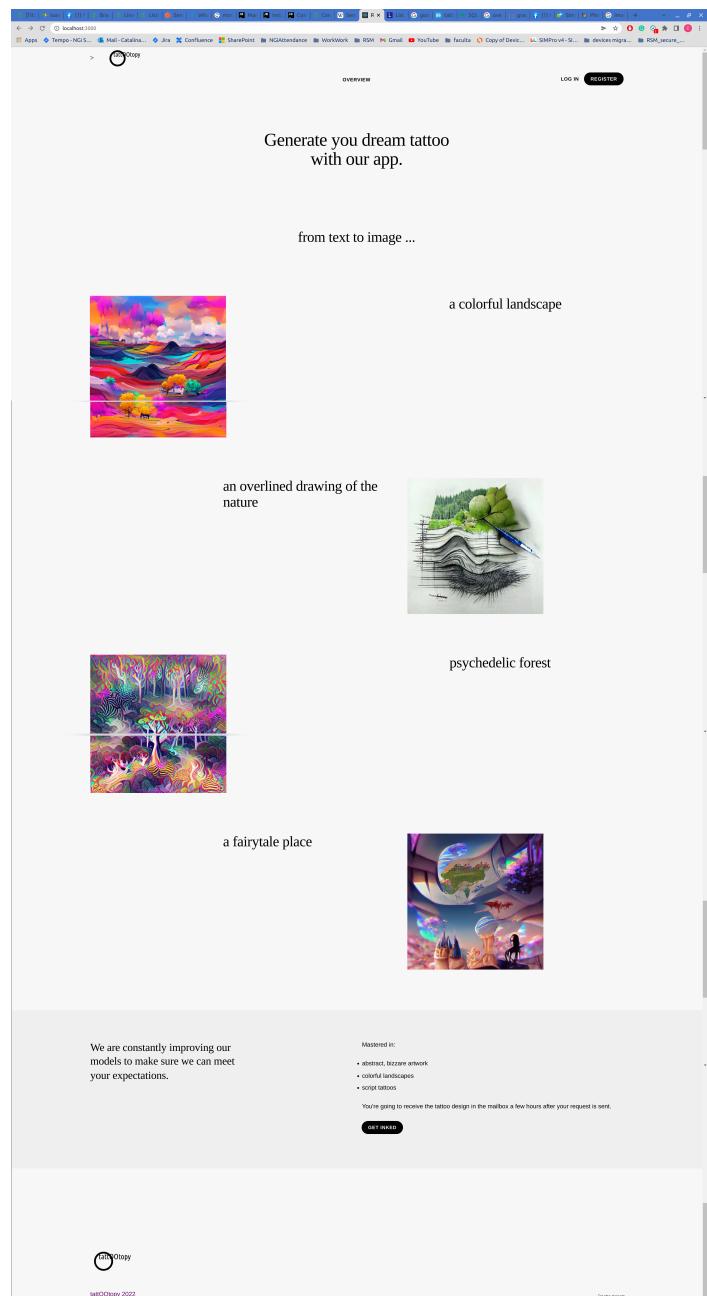


Figura 6.4: Pagina de Overview

Utilizatorii care nu au un cont pe platformă trebuie să își creeze unul înainte



Figura 6.5: Logo-ul aplicației

de a putea folosi funcționalitățile acesteia. La apăsarea butonului de **Register** se deschide pagina 6.6. Aici, utilizatorul completează formularul cu datele: adresa de email, parolă, prenume și nume. Formularul oferă posibilitatea autocompletării adresei de email, a prenumelui și a numelui în funcție de datele salvate în browser 6.7.

Figura 6.6: Pagina pentru crearea unui cont nou

După completarea câmpurilor, utilizatorul apasă butonul **register**. Pentru câmpurile necompletate se afișează un warning 6.8, pentru câmpul **email** se verifică formatul, iar pentru câmpul **password** se verifică cât de puternică este parola. În cazul în care adresa de email și/sau parola nu îndeplinesc cerințele se

Figura 6.7: Autocompletarea câmpurilor din form-ul de Registrare

Figura 6.8: Warning pentru completarea necorespunzătoare a câmpurilor - Register Form

afisează o eroare 6.13. Dacă toate câmpurile au fost completate corespunzător, utilizatorul este redirectionat către pagina **Home**.

Adresa de email are un format valid dacă este de forma ***@[domain]. Parola are un format valid dacă conține cel puțin: 8 caractere, o majusculă, o cifră și un caracter special.

În cazul în care utilizatorul are deja un cont pe platformă, acesta se poate autentifica prin apăsarea butonului **Login**. Utilizatorul va fi redirectionat către pagina 6.9, unde poate completa credențialele sale (adresa de email și parola). Își în acest form se poate face autocompletarea datelor dacă acestea au fost salvate în browser 6.10. Vizualizarea parolei sau ascunderea acesteia se poate face prin apăsarea butonului de **Show password** 6.11.

După completarea câmpurilor, se apasă butonul de **login**. Dacă adresa de email și parola au fost completate corespunzător și serverul validează autentificarea, utilizatorul este redirectionat către pagina de **Home**. În cazul în care cel puțin unul dintre câmpuri este necompletat, se afișează un warning 6.12.

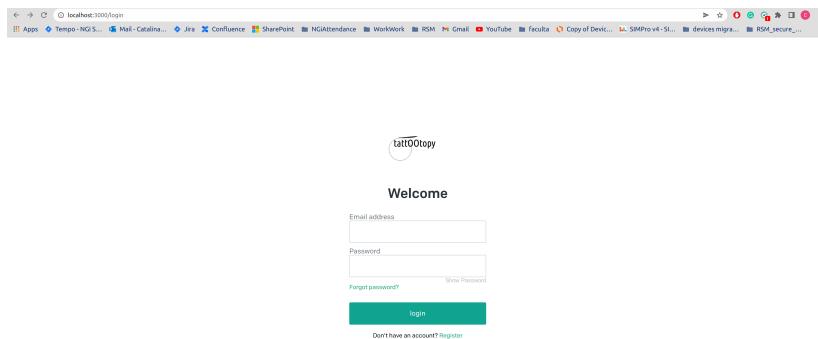


Figura 6.9: Pagina de autentificare

Utilizatorul poate naviga între paginile de **Login** și **Register** prin apăsarea hyperlink-urilor *Register*, respectiv *Login*.

După autentificare, utilizatorul este redirectionat către pagina **Home** 6.14. Aici utilizatorul poate folosi funcționalitatea de generare a unui tatuaj nou prin completarea formularului *Try me*. Formarul trebuie completat cu o descriere de tipul ("a nice, happy tree with purple and red leaves") în câmpul input și, optional, cu o imagine din propriul computer în secțiunea *Choose file*.

Mai jos în pagina este afișată lista de tatuaje generate anterior de către utilizatorul autentificat. Lista de entități este afișată la încărcarea paginii (se trimit un Request către serverul Java la deschiderea paginii). Pentru fiecare item există două butoane: **make public** și **delete**. Butonul toggle de **make public** are două stări (on și off) care reprezintă că imaginea este publică, respectiv că imaginea este privată. Butonul de **delete** șterge tatuajul generat și acesta nu va mai apărea în lista utilizatorului.

Butonul de **Logout** redirectionează utilizatorul către pagina de **Overview** 6.4.



Welcome

Email address

Password
 sapca.catalina@gmail.com
sapca.catalina@yahoo.com

[Forgot password?](#) [Manage...](#)

Don't have an account? [Register](#)



Welcome

Email address
 sapca.catalina@gmail.com

Password
 aaa

[Forgot password?](#) [Show Password](#)

Don't have an account? [Register](#)

Figura 6.10: Autocompletarea câmpurilor din form-ul de Login

Figura 6.11: Buton de ascundere/vizualizare a parolei - Login Form



Welcome

Email address

Password
 ! Please fill out this field.

[Forgot password?](#) [Show Password](#)

Don't have an account? [Register](#)



Create Your Account

Email address
 aaaaa
wrong email format!

First name
 jh

Last name
 i

Password

weak password

Already have an account? [Log in](#)

Figura 6.12: Warning pentru completarea necorespunzătoare a câmpurilor - Login Form

Figura 6.13: Eroare format invalid - Register Form

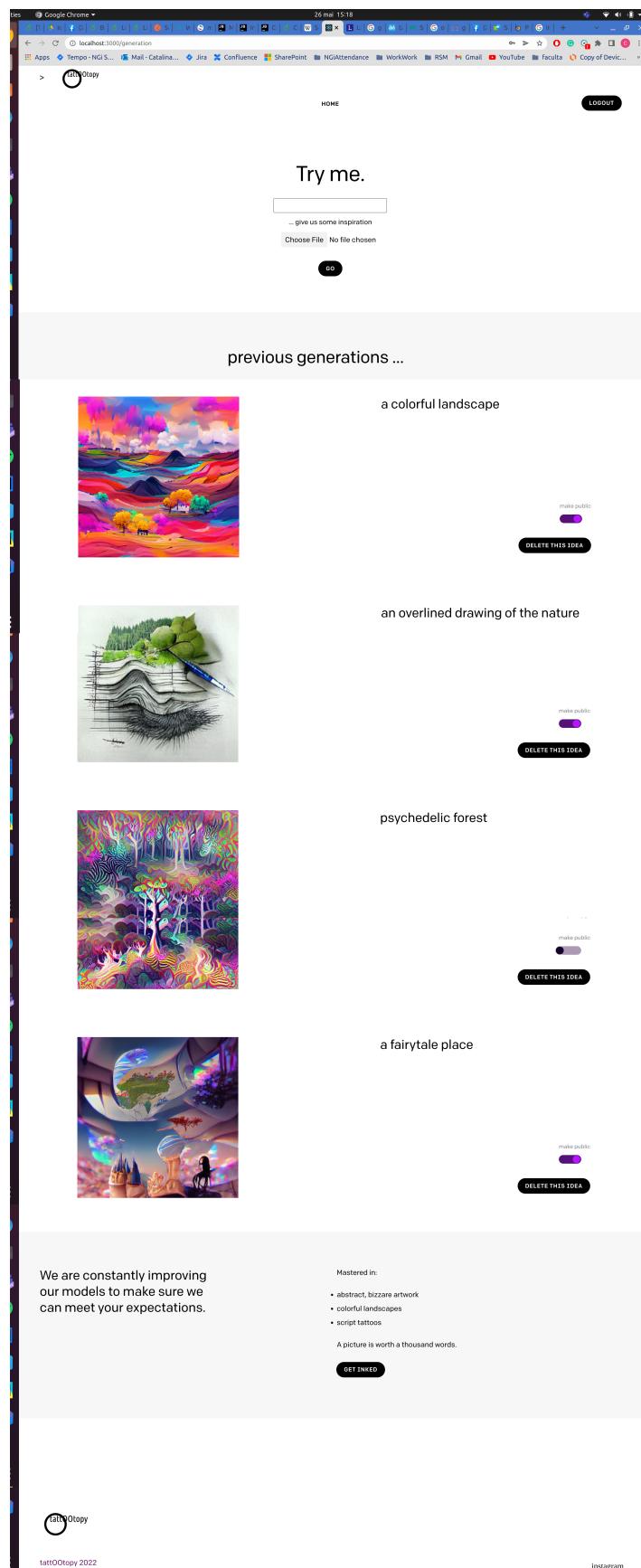


Figura 6.14: Pagina de autentificare

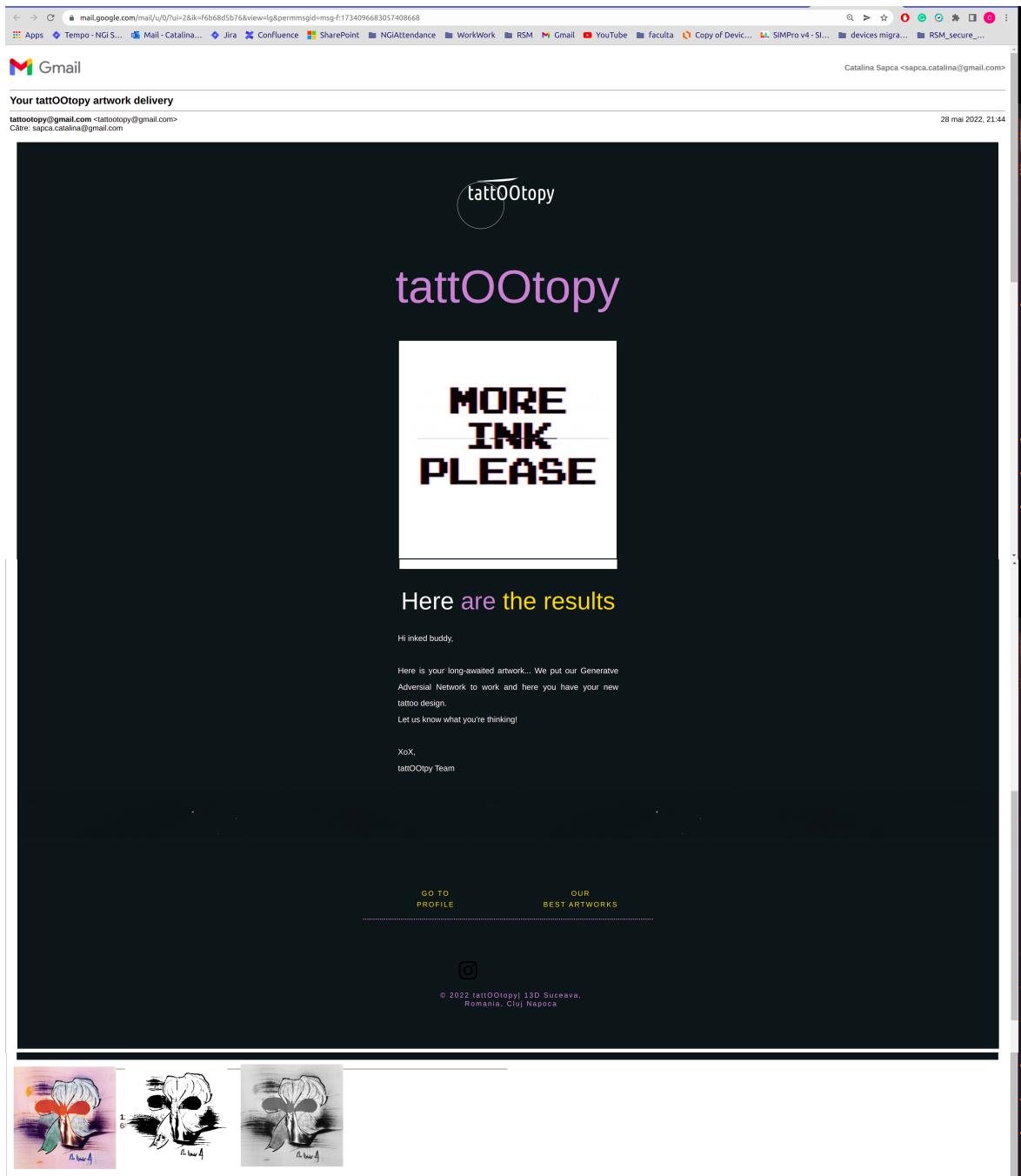


Figura 6.15: Corpul email-ului

Capitolul 7

Concluzii și dezvoltări ulterioare

Lucrarea a reușit să îmbunătățească modelul VQGAN-CLIP [CBK⁺22] prin adăugarea componentei lingvistice de generare a descrierilor echivalente și a dovedit obținerea rezultatelor concludente în urma testelor efectuate. Componenta de adăugare de filtre aduce soluția mai aproape de ideea de design de tatuaj, creând un blueprint pentru client sau pentru artistul tatuator. Aplicația dezvoltată îmbracă soluția într-o interfață user-friendly și oferă utilizatorilor o metodă de gestionare a generărilor.

Cu toate acestea, există direcții viitoare de dezvoltare ale aplicației și puncte care nu au fost atinse în lucrarea prezenta, după cum urmează 7.1.

7.1 Dezvoltări ulterioare

O îmbunătățire semnificativă a modului de funcționare al aplicației este fine-tuning-ul modelului folosit pe mai multe date de tip imagini cu tatuaje. Acest lucru nu a fost posibil datorită faptului că fine-tuning-ul presupune colectarea și etichetarea unui număr foarte mare de date ceea ce ar fi fost foarte costisitor din punct de vedere al timpului. Momentan nu există niciun dataset exclusiv cu imagini cu tatuaje care ar fi putut fi folosit în antrenarea modelului de VQGAN.

Platforma poate fi îmbunătățită și extinsă cu o pagină de search. Această pagină de search permite căutarea imaginilor după cuvinte cheie (regăsite în descrierile care au generat aceste imagini). Astfel, utilizatorii pot vizualiza și accesa imaginile generate de către alți utilizatori, dacă acestia au ales să le facă publice (prin setarea atributului *is Public* al unei generări). Această extensie este ușor realizabilă pe viitor întrucât toate datele necesare unei căutări după descriere sunt persistante în baza de date.

Bibliografie

- [Bar13] Kim Barbour. It can be quite difficult to have your creativity on tap: balancing client expectations and artistic practice in the tattoo industry. In *Projections-Proceedings of WCCA'2013-VI World Congress on Communication and Arts*, pages 2–7. COPEC–Science and Education Research Council, 2013.
- [Ble]
- [BS18] Shane Barratt and Rishi Sharma. A note on the inception score, 2018.
- [CBK⁺22] Katherine Crowson, Stella Biderman, Daniel Kornis, Dashiell Stander, Eric Hallahan, Louis Castricato, and Edward Raff. Vqgan-clip: Open domain image generation and editing with natural language guidance, 2022.
- [DYH⁺21] Ming Ding, Zhuoyi Yang, Wenyi Hong, Wendi Zheng, Chang Zhou, Da Yin, Junyang Lin, Xu Zou, Zhou Shao, Hongxia Yang, and Jie Tang. Cogview: Mastering text-to-image generation via transformers, 2021.
- [ERO20] Patrick Esser, Robin Rombach, and Björn Ommer. Taming transformers for high-resolution image synthesis, 2020.
- [GLY21] Theodoros Galanos, Antonios Liapis, and Georgios N. Yannakakis. Affectgan: Affect-based generative art driven by semantics, 2021.
- [Imp]
- [Jet]
- [Mul]
- [NDR⁺21] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models, 2021.
- [PNDA21] Md Aminul Haque Palash, Md Abdullah Al Nasim, Aditi Dhali, and Faria Afrin. Fine-grained image generation from bangla text description using attentional generative adversarial network, 2021.

- [RAY⁺16] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis, 2016.
- [RDN⁺22] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents, 2022.
- [RKH⁺21] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021.
- [RN66] A. Michael RNoll. Human or machine: A subjective comparison of piet mondrian’s “composition with lines” (1917) and a computer-generated picture. *The Psychological Record*, 16(1), 1966.
- [RPG⁺21] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation, 2021.
- [RZZ⁺21] Shulan Ruan, Yong Zhang, Kun Zhang, Yanbo Fan, Fan Tang, Qi Liu, and Enhong Chen. Dae-gan: Dynamic aspect-aware gan for text-to-image synthesis, 2021.
- [SKB21] Chiheon Kim Doyup Lee Saehoon Kim, Sanghun Cho and Woonhyuk Baek. mindall-e on conceptual captions. <https://github.com/kakaobrain/minDALL-E>, 2021.
- [SWR20] Douglas M. Souza, Jônatas Wehrmann, and Duncan D. Ruiz. Efficient neural architecture for text-to-image synthesis, 2020.
- [XZH⁺17] Tao Xu, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang, and Xiaodong He. AttnGAN: Fine-grained text to image generation with attentional generative adversarial networks, 2017.
- [YWL⁺20] Chao Yang, Guoqing Wang, Dongsheng Li, Huawei Shen, Su Feng, and Bin Jiang. Ppgn: Phrase-guided proposal generation network for referring expression comprehension, 2020.
- [YZD21] Yu Yu, Weibin Zhang, and Yun Deng. Frechet inception distance (fid) for evaluating gans, 09 2021.
- [ZPCY19] Minfeng Zhu, Pingbo Pan, Wei Chen, and Yi Yang. Dm-gan: Dynamic memory generative adversarial networks for text-to-image synthesis, 2019.
- [ZS21] Zhenxing Zhang and Lambert Schomaker. DTGAN: Dual attention generative adversarial networks for text-to-image generation. In *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, jul 2021.

- [ZXL⁺16] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks, 2016.
- [ZXL⁺17] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris Metaxas. Stackgan++: Realistic image synthesis with stacked generative adversarial networks, 2017.
- [ZZC⁺21] Yufan Zhou, Ruiyi Zhang, Changyou Chen, Chunyuan Li, Chris Tensmeyer, Tong Yu, Jiuxiang Gu, Jinhui Xu, and Tong Sun. Lafite: Towards language-free training for text-to-image generation, 2021.