Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Руксова Євгенія Вікторовича*
(ПІБ)

академічної групи *122-19ск-1*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Методи і моделі для реалізації концепції машинного мистецтва*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*проф. Мороз Б.І.*			
розділів:				
спеціальний	*проф. Мороз Б.І.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент	*проф. Корнієнко В.І.*			
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2022

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 2022 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-19ск-1* *Руксова Є. В.*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Методи і моделі для реалізації концепції машинного мистецтва*

затверджена наказом ректора НТУ «ДП» від « » червня 2022 р. №

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2022 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2022 р.***

Завдання видав _____ *проф. Мороз Б. І.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Руксов Є. В.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2022 р.*

Термін подання кваліфікаційної роботи до ЕК: *13.06.2022 р.*

РЕФЕРАТ

Пояснювальна записка: 120 с., 18 рис., 3 табл., 4 дод., 22 джерела.

Об’єкт розробки: інформаційна система для тренування та тестування генеративно-змагальної нейронної мережі.

Мета кваліфікаційної роботи: реалізація інформаційної системи, яка дозволяє тренувати та тестувати генеративно-змагальну нейронну мережу, завантажувати та зберігати набори тренувальних даних, зберігати та використовувати навчені стани моделі.

У вступі надається аналіз сучасного стану проблеми та напрямки досліджень в цій сфері, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми.

У першому розділі проведено аналіз предметної галузі, визначено актуальність та призначення розробки, розроблена детальна постановка завдання, вказані вимоги до програмної реалізації, технологій, програмних засобів, безпеки та сумісності розробленої системи.

У другому розділі виконано аналіз існуючих рішень, обрано методологію розробки, описано основні аспекти проектування і розробка програми, наведено опис алгоритмів і структур функціональних модулів системи, визначені вхідні і вихідні дані, наведені характеристики технічних засобів, описаний процес виклику та завантаження програми, описано інтерфейс взаємодії з системою.

В економічному розділі визначено трудомісткість розробки інформаційної системи, проведений підрахунок вартості створення програми та розраховано час на його реалізацію.

Практичне значення проекту полягає в вирішенні завдання розробки інформаційної системи, яка дозволяє тренувати генеративно-змагальну модель нейронної мережі на завантажених користувачем наборах графічних даних та тестувати роботу навчених на цих наборах станів моделі.

Актуальність розробленої системи полягає в тому, що ця система реалізує одну з найновітніших моделей машинного навчання, яка надала значного розвитку концепції машинної творчості, що, в свою чергу, дозволяє більш ефективно вивчати феномен творчості, як такий. Також актуальним є застосування для вирішення поставленої задачі сервісу хмарних обчислень, що значно розширює можливості для ефективного впровадження таких систем.

Список ключових слів: МАШИННА ТВОРЧІСТЬ, МАШИННЕ НАВЧАННЯ, ГЕНЕРАТИВНО-ЗМАГАЛЬНА НЕЙРОННА МЕРЕЖА, СЕРВІС ХМАРНИХ ОБЧИСЛЕНЬ, AWS, SAGEMAKER, JULIA, FLUX, JAVA, SPRING BOOT, SWAGGER.

ABSTRACT

Explanatory note: 120 pages, 18 figures, 3 tables, 4 appendices, 22 sources.

Object of development: information system for training and testing of generative adversarial neural network.

The purpose of the qualification work: the implementation of an information system that allows you to train and test the generative adversarial neural network, download and store training data sets, store and use the trained states of the model.

The introduction provides an analysis of the current state of the problem and researches in this area, specifies the purpose of the qualification work and the field of its application, provides a justification for the relevance of the topic.

The first section analyzes the subject area, determines the relevance and purpose of development, developed a detailed statement of the task, the requirements for software implementation, technology, software, security and compatibility of the created system.

The second section analyzes the existing solutions, selects the development methodology, describes the main aspects of design and implementation of the program, describes the algorithms and structures of functional modules of the system, defines input and output data, characteristics of technical stuff, describes the process of calling and loading the program, defines the interface of interaction with the system.

The economic section determines the complexity of information system development, calculates the cost of creating a program and calculates the time for its implementation.

The practical significance of the project is to solve the problem of developing an information system that allows you to train the generative adversarial model of a neural network on uploaded by user graphic data sets and test the work of model states trained on these sets.

The relevance of the developed system is that this system implements one of the latest models of machine learning, which has significantly developed the concept of machine creativity, which, in turn, allows more effective study of the phenomenon of creativity as such. Also relevant is the use of cloud computing service to solve the problem, which significantly expands the possibilities for the effective implementation of such systems.

Keywords: MACHINE CREATIVITY, MACHINE LEARNING, GENERATIVE ADVERSARIAL NEURAL NETWORK, CLOUD COMPUTING, AWS, SAGEMAKER, JULIA, FLUX, JAVA, SPRING BOOT, SWAGGER.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

GAN – генеративно-змагальна нейронна мережа;

DCGAN – глибока згорткова генеративно-змагальна нейронна мережа;

CNN – згорткова нейронна мережа;

AWS – сервіс хмарних обчислень компанії Amazon;

ПЗ – програмне забезпечення;

БД – база даних;

ООП – об’єктно-орієнтоване програмування;

JVM – віртуальна машина Java;

IoC – шаблон проектування «інверсія контролю»;

NoSQL – нереляційні бази даних;

UML – універсальна графічна мова для моделювання та проектування.

ВСТУП

Відома теза «мозок людини – найскладніша обчислювальна система в світі» має багато підтверджень та проявів, особливо, у сучасному світі. Існує певний набір показників, значення яких свідчить про існування та рівень розвитку цивілізації. Мозок, як велика нейронна мережа, якість і швидкість навчання якої та складність оброблених та згенерованих нею структур надзвичайно високі, зміг дійти до такого рівня ефективності, що зі складних структур навколишнього світу зміг створити ще складніші. Саме це і є головною ознакою, яка характеризує цивілізацію.

Проте такий комплексний процес, як творчість та мистецтво, дуже складно оцінити або проаналізувати. Складною задачею є визначення механізму, який реалізує процес синтезу чогось нового. Над цією задачею працювало багато вчених, філософів та митців. Бажання пізнавати в людині не менше ніж бажання творити, тому спроби пізнати саму сутність творчості з наукової точки зору робляться постійно. Певного успіху в цьому напрямку досягли саме математики та спеціалісти з інформаційних технологій. Вони пішли шляхом моделювання найбільш близького до людського мозку механізму, який синтезує складну систематизовану інформацію. Розроблювана система призначена для синтезу інформації саме графічного типу.

Ще однією ключовою ідеєю, яка покладена в основу цієї інформаційної системи, є взаємне навчання різних нейронних мереж. Як відомо, люди часто застосовують імітаційний підхід. Розробляючи модель, яка відображає певний процес або явище, науковець може несподівано створити те, що отримає набагато ширше застосування, аніж він сам розраховував. У випадку із взаємним навчанням, напевно так і вийшло. Відомо, що для ефективної роботи штучної нейронної мережі її треба спочатку навчити, тобто адаптувати під вирішення певного завдання. Цю ідею реального світу давно взяли за основу в сфері машинного навчання, але також існує й феномен взаємного навчання, наприклад, двох іноземців, які вивчають мову один одного. Вони в процесі спілкування можуть знаходити помилки один одного та вказувати на них, тим самим

покращуючи свою майстерність спілкування на цій мові. Цю ж концепцію вирішив реалізувати у вигляді математичної моделі та відповідного алгоритму навчання Ян Гудфеллоу в 2014 році. В результаті ця модель стала майже єдиною дійсно ефективною для реалізації саме концепції машинної творчості, що зробило великий внесок у розвиток сфери досліджень феномену творчості.

Ще одним аспектом роботи є застосування сервісу хмарних обчислень AWS, який має неабияку популярність серед сучасних IT-компаній, через свою зручність використання та відносно невелику ціну за користування. Крім того, на базі цього хмарного сервісу можна розгорнути таку апаратну конфігурацію, яка дозволяє швидко та ефективно виконувати навчання складних моделей.

Розроблювана система має перспективу застосування в сфері досліджень описаного явища. Так як це зовсім молода концепція, все ще існує багато чого, що може бути винайдено та досліджено. Зокрема, ця інформаційна система дозволяє експериментувати із різними наборами даних для навчання та різною кількістю циклів навчання, тож користувач може знайти якусь цікаву конфігурацію. Крім наукових досліджень для побудови теоретичної системи опису феномену мистецтва, результати цієї роботи будуть корисними і для спеціалістів з аналізу даних, адже розглянута модель вміє не тільки генерувати зображення, а й розпізнавати справжні та фальшиві зображення (проте запуск моделі саме на розпізнавання, а не на генерування, в розробленій система виконується вручну). Це може знадобитися, наприклад, в аналізі великої бази картин, і знаходження справжніх, які належать пензлю того чи іншого митця. Також є потенціал застосування такої моделі, наприклад, при розробці ігор, для генерування унікальних ігрових об’єктів. Отже, актуальність цієї роботи має широке значення, як практичне, так і теоретичне та фундаментальне.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості з предметної галузі

Описана система відноситься до інформаційних систем побудованих на базі математичної моделі класу машинного навчання. Для розуміння сутності машинного навчання необхідно дати визначення деяким термінам, зокрема алгоритм навчання, нейронна мережа. Також треба окреслити типи задач, які можуть бути ефективно вирішені цими алгоритмами й моделями.

Почнемо з визначення того, що таке алгоритм навчання. Машинне навчання, по суті, є специфічною методологією із сфери прикладної статистики з використанням обчислювальних машин для статистичної оцінки складних функцій і доказу довірчих інтервалів навколо цих функцій. Алгоритм навчання – це алгоритм, який може вчитися на певних наборах вхідних даних. Але що ми маємо на увазі під навчанням? Том Мітчелл у своїй книзі дає стисле визначення: «Комп'ютерна програма навчається на досвіді E у відповідності до деякого класу завдань T та з показником ефективності P тоді й тільки тоді, коли її ефективність P при виконанні завдань із класу T покращується з досвідом E» [1]. Можна уявити широкий спектр видів досвіду E, завдань T і показників продуктивності P. Для наочного розуміння сутності цього поняття наведемо деякий приклад: комп'ютерна програма, яка намагається грати в шашки, може покращити свою ефективність, виміряну в її здібності вигравати в класі завдань, пов'язаних із грою в шашки, завдяки досвіду, отриманому під час гри проти себе – така програма визнається такою, що навчається. Машинне навчання – це напрям теорії статистики у поєднанні з теорією алгоритмів, який займається вивчанням та створенням алгоритмів навчання та їх реалізації у вигляді комп'ютерних програм для вирішення певних задач з теорії статистики.

Тепер розглянемо кожен з елементів визначення, яке дав Том Мітчелл:

1. Завдання машинного навчання T зазвичай описуються з точки зору того, як система повинна обробляти вхідні дані. Це, зазвичай, набір ознак, які

були кількісно виміряні з певного об’єкта чи події, які ми хочемо, щоб система обробляла. Наприклад, ознаками зображення можуть бути значення яскравості пікселів, і ці данні зібрані у матрицю заданої розмірності. За допомогою алгоритмів машинного навчання можна вирішити багато видів завдань. Ось деякі з найпоширеніших завдань із сфери статистики, які доволі ефективно вирішують алгоритми машинного навчання: класифікація, кластеризація, регресія, транскрипція (поверхневий опис неструктурованих даних), машинний переклад, структурований вивід (структурування даних), виявлення аномалій, придушення шумів, синтез і вибірка (сюди ж відноситься концепція машинного мистецтва, яка покладена в основу цієї роботи), доповнення відсутніх значень. Звичайно, цей список неповний і таких задач набагато більше.

2. Щоб оцінити якість алгоритму навчання, ми повинні розробити кількісну оцінку його ефективності. Зазвичай цей показник ефективності P є специфічним для завдання T, яке виконує система. Для таких завдань, як класифікація, кластеризація, транскрипція, ми часто вимірюємо точність моделі. Точність – це кількість примірників, для яких модель видає правильний результат, поділено на загальну кількість примірників. Ми також можемо отримати еквівалентну оцінку, вимірюючи частоту помилкових висновків моделі. Зазвичай, ця оцінка має значення від 0 до 1. Також часто моделі дають ймовірнісний результат, тоді це потребує більш складного принципу оцінки, наприклад, обчислення взаємної або відносної ентропії розподілення ймовірностей. На додачу, нас ще цікавить, наскільки добре алгоритм навчання працює з даними, які він раніше не бачив, оскільки це визначає, наскільки добре він працюватиме при використанні в реальному світі. Тому ми оцінюємо ці показники ефективності, використовуючи тестові набори даних, які формуються окремою від даних, які використовуються для навчання моделі. Вибір способу оцінки ефективності може здатися простим і об’єктивним, але часто буває важко обрати такий спосіб оцінки, який би добре зіставляв бажану поведінку системи та реальну і давав достатньо репрезентативне чисельне представлення

результату цього порівняння. Тож при побудові системи машинного навчання необхідно підбирати цей параметр виходячи від специфіки поставленої задачі.

3. З точки зору способу набуття досвіду E алгоритмом навчання їх можна загалом класифікувати на навчання з вчителем та без вчителя залежно від того, яка інформація відома про дані з тренувального набору під час навчання. Одним із найстаріших наборів даних, які вивчали математики та дослідники машинного навчання, є набір даних Iris (1936 року), це збірка вимірювань різних частин 150 підвидів квітки ірис [2]. Характеристиками кожного примірника є заміри кожної частини рослини: довжина чашолистка, ширина чашолистка, довжина пелюстки та ширина пелюстки. В наборі даних також зафіксовано, до якого виду належала кожна рослина. Алгоритми навчання без вчителя отримують досвід від набору даних, що містить багато параметрів, визначаючи корисні властивості структури цього набору для подальшого формування висновку. Ми зазвичай ставимо на меті вивчити весь розподіл варіативностей, який має тренувальний набір, або явним способом, як при класифікації, або неявно, як при синтезі чи видаленні шумів. Алгоритми навчання з вчителем мають набір даних, що містить параметри, але кожному примірнику відповідає мітка або очікуваний результат обробки. Наприклад, в навчальному наборі Iris кожен примірник позначений назвою певного виду рослини. Алгоритм навчання з вчителем може вивчати набір Iris і навчатися класифікувати рослини на відповідні види на основі їх параметрів. Також існує третій спосіб набуття досвіду – навчання з підкріпленням, суть якого полягає в оцінці якості роботи моделі від навколишнього середовища. Навколишнім середовищем може виступати якась інша система, яка має свій внутрішній алгоритм реагування на дії моделі. Це, як правило, алгоритми, які є основою програмного забезпечення для роботів.

Розроблювана модель відноситься до глибокого навчання, сутність якого полягає в побудові комплексної моделі машинного навчання, яка включає в себе різні методи вирішення завдання. В даному випадку ми кажемо про модель, яка одночасно має декілька з описаних характеристик і одразу відноситься до кількох типів моделі в класифікації. Зокрема, в роботі розглядається

генеративно-змагальна нейрона мережа, про яку докладніше буде викладено в другому розділі.

Для повноти розуміння сутності цієї інформаційної системи варто згадати про таке поняття як машинне мистецтво. Комп'ютерна творчість (також відома як штучна творчість, машинна творчість, креативні обчислення, творчі обчислення, машинне мистецтво) – це багатопрофільна діяльність, яка знаходиться на перетині сфер штучного інтелекту, когнітивної психології, філософії та мистецтва.

Метою обчислювальної творчості є моделювання або імітація творчості за допомогою комп'ютера для досягнення однієї з кількох цілей:

– створити програму, яка здатна до творчості на рівні людини;

– щоб краще зрозуміти творчість людини та сформулювати алгоритмічний погляд на творчу поведінку людей;

– розробляти програми, які можуть підвищити творчу активність людини, не обов'язково при цьому мати високу креативність [3].

Сфера обчислювальної творчості стосується теоретичних і практичних питань у вивченні творчості. Теоретична робота щодо природи та правильного визначення креативності виконується паралельно з практичною роботою над впровадженням систем, що демонструють креативність, при цьому один напрям роботи доповнює інший. Проте ця робота направлена на інше, а саме аналіз конкретних методів та принципів, які дозволяють реалізувати практичну сторону машинної творчості.

Ще один аспект, який був розглянутий та застосований при розробці описаної інформаційної системи, – сервіси хмарних обчислень.

На поточному етапі розвитку комп'ютерних технологій має місце тенденція до зростання попиту на хмарні сервіси. Це обумовлено тим, що об'єм та складність обчислень досягли такого рівня, що використання власних серверів та обчислювальних систем стало занадто дорого. Висока ціна обумовлена необхідністю постійного неперервного обслуговування великої обчислювальної мережі, тобто треба постійно мати великий штат системних адміністраторів, які

будуть слідкувати та підтримувати мережу у працездатному стані, і це на додачу до штату програмістів та DevOps. Багато IT-компаній, шукаючи альтернативи, прийшли до хмарних сервісів.

Хмарні обчислення базуються на принципі доступності ресурсів комп'ютерної системи, особливо сховища даних (хмарного сховища) та обчислювальної потужності тільки тоді, коли це необхідно, та без прямого активного керування користувачем внутрішньою інфраструктурою. Великі хмари часто мають сервіси, розподілені в кількох місцях, кожне місце є центром обробки даних. Хмарні обчислення покладаються на спільне використання ресурсів для досягнення узгодженості та зазвичай використовують модель «оплати по мірі використання», яка може допомогти зменшити капітальні витрати, але також може призвести до неочікуваних операційних витрат для необізнаних пересічних користувачів. Проте загалом для побудови великої та складної обчислювальної системи хмари набагато дешевші та мають більш простий інтерфейс для налагодження та підтримки, що може звільнити IT-компанію від необхідності тримати великий штат системних адміністраторів.

Ще однією причиною використання хмар є те, що ці сервіси надають можливість запускати алгоритми машинного навчання на базі обчислювальних платформ різного типу та потужності – а отже можна порівняти ефективність різних конфігурацій системи для навчання описаної моделі і зробити вибір.

Одним з найпопулярніших хмарних сервісів є AWS (Amazon Web Services). Це дочірня компанія Amazon, яка надає приватним особам, компаніям і урядам платформи хмарних обчислень за принципом «коли використовуєш, тоді платиш» також надає API на основі фіксованої оплати, ступінь користування кожною послугою при цьому вимірюється по-різному, а тому різні сервіси мають різний вклад в загальну вартість користування хмарою AWS. Для досягнення мети даної роботи буде використовуватись саме ця платформа, адже кількість наявної у відкритому доступі документації та якість служби підтримки достатньо висока, що й каже про широку розповсюдженість саме AWS серед IT-компаній.

1.2. Призначення розробки та галузь застосування

Розроблюється система, яка реалізує концепцію машинного графічного мистецтва через самонавчальну модель генеративно-змагальної нейронної мережі із згортковими шарами. Крім безпосередньої побудови моделі, реалізується також алгоритм її навчання, механізм нормалізації вхідної вибірки даних та спосіб зберігання стану моделі для подальшого тестування. Всі ці елементи реалізовані на мові програмування Julia. На додачу, ця модель інтегрована із розробленою в рамках цієї роботи системою керування та зберігання наборів даних та станів моделі. Система керування є back-end застосунком, який має зовнішній інтерфейс, через який користувач або інша система може керувати та зберігати набори даних, а також виконувати тестування навченої моделі. Ця частина системи реалізована на мові програмування Java з використанням фреймворку Spring. Вся система побудована на базі сервісу хмарних обчислень AWS.

Система має такі галузі застосування як машинне мистецтво та обробка зображень. Перша галузь має два пов’язаних з розроблюваною системою напрямки розвитку: науково-дослідницький в сфері когнітивних наук та створення унікальних об’єктів в комп’ютерних іграх. Для другої галузі ця система може надати такі можливості як: генерування зображень у певному стилі або з певним сюжетом, розпізнання фальшивих зображень. Загалом цю система можна розгортати в рамках великих програмних модулів, які займаються генеруванням та обробкою графічних даних. Або для різних досліджень та експериментів в рамках вивчення цивілізаційного феномену творчості та образотворчого мистецтва.

1.3. Підстави для розробки

Підставою для розробки кваліфікаційної роботи на тему «Методи і моделі для реалізації концепції машинного мистецтва» є наказ ректора по НТУ «Дніпровська політехніка» від __.__. 2022 р. №____.

1.4. Постановка завдання

Неможливо уявити сучасні наукові дослідження без математичних моделей, на основі яких створюється теоретична основа для багатьох напрямків досліджень. Проте на поточному рівні розвитку науки математичні моделі неможливо розраховувати без застосування обчислювальних машин. Великі складні моделі, які відображають складні об’єкти реального світу, потребують мільйони операцій, щоб розрахувати різні параметри і в результаті дослідник міг робити висновки в своїх роботах.

До таких складних моделей відноситься мозок людини, який надихнув математиків та спеціалістів з інформаційних технологій створити модель, яка здатна навчатися виконувати різні задачі. Задачі, в яких використовуються моделі машинного навчання, зазвичай не мають простого чітко визначеного детермінованого алгоритму для вирішення. Тобто ці задачі відносяться до класу складності NP (задачі, які можна вирішити недетермінованим алгоритмом за поліноміальний час). Задача, яка розглядається в цій роботі, відноситься до класу задача визначення закономірностей та зв’язків у певному наборі зображень, щоб потім згенерувати зображення, яке буде мати таку саму закономірність як представлений набір, тобто будуть класифіковані так само. Додаткова складність цієї задачі полягає в тому, що насправді тут є дві нетривіальні задачі, які треба вирішувати паралельно і в комплексі:

- класифікація зображення на справжні та підробки;
- генерація зображення із випадкового вхідного набору даних (шуму).

Для вирішення цієї комплексної задачі Яном Гудфелоу була розроблена модель генеративно-змагальної нейронної мережі. Більш докладно про структуру та принцип роботи цієї моделі буде зазначено в другому розділі пояснювальної записки. Також були розроблені різні методики оптимізації процесу навчання, що також буде докладно зазначено далі.

Доцільність розробки такої моделі для вирішення поставлених задач була доведена численними експериментами та тестами. Серед всіх існуючих зараз

моделей машинного навчання для вирішення такого типу задач генеративно-змагальна нейронна мережа найбільш ефективна, а її швидкодія на базі певного типу обчислювальних машин достатня для зручного застосування користувачем.

Крім безпосередньо моделі та алгоритму навчання, в системі ще є модулі, які забезпечують роботу з наборами даних для навчання та реалізують інтерфейс для тестування моделі. Ці модулі виконують обслуговуючу функцію для нейронної мережі. В багатьох інших подібних системах нейронна мережі або модель машинного навчання є одним із додаткових елементів, який підвищує «інтелектуальність» програми. Проте в таких системах нейронні мережі зазвичай вже навчені завчасно і лише експлуатуються. Запропонована ж система побудована навколо нейронної мережі й основні задачі – саме обслуговування та створення бази таких навчених моделей на різних наборах даних, щоб в подальшому використовувати ці готові моделі там, де це необхідно.

Кінцевий користувач цієї системи – аналітик даних, який займається збором та систематизацією графічної інформації та будує на її основі генеративні моделі. Вхідними даними для системи є набори зображень та їх метадані. Вихідними об’єктами є навчені моделі, користувачами яких є розробники такого програмного забезпечення, яке потребує генерації унікальних зображень, розпізнання підробок.

Деякі етапи загального процесу користування системою не є автоматичними через певні технічні обмеженні. Зокрема введення навченої моделі в експлуатацію іншою системою є ручним.

Отже, метою роботи є створення системи, яка будує та навчає генеративно-змагальну модель нейронної мережі, нормалізує вхідні дані для навчання, зберігає набори даних та стани моделі після навчання. Для досягнення поставленої мети необхідно виконати наступні задачі:

– здійснити аналіз предметної галузі з метою виявлення сучасного стану обраної проблематики;

– здійснити аналіз існуючих рішень щодо побудови зазначеної моделі;

– здійснити аналіз методів оптимізації процесу навчання моделі;

– вибрати інструментальні засоби для розробки алгоритму моделі;

– розробити алгоритм нормалізації вхідних даних, побудови та навчання моделі, збереження стану моделі;

– розробити модулі обслуговування моделі та визначити механізми їх взаємодії між собою;

– розробити зрозумілий інтерфейс-документацію для користувача та розробника тієї системи, яка буде використовувати даний програмний модуль.

1.5. Вимоги до програми або програмного виробу

Фундаментальними критеріями якості для будь-якого програмного продукту є висока оцінка роботи системи від користувача. Так як ця система є back-end програмою, то в якості користувача виступає інша програма або програмний модуль, який спілкується із системою через певний інтерфейс. Також користувачем системи є розробники та аналітики даних, які працюють над створенням своїх застосунків, які будуть взаємодіяти з описаною системою. Тому виходячи з того, хто саме є користувачем системи будується список вимог до програми. Звісно, зазначені нижче вимоги можуть змінюватися в процесі подальшої модернізації системи та розширення функціоналу.

1.5.1. Вимоги до функціональних характеристик

Для досягнення поставленої мети в рамках системи повинні бути реалізовані такі можливості:

– завантажувати набори графічних даних для навчання моделі в систему;

– запуск процесу навчання моделі на основі набору даних;

– отримання поточного стану моделі після навчання;

– тестування моделі.

1.5.2. Вимоги до інформаційної безпеки

Для уникнення проблем під час роботи із системою необхідно реалізувати:

– обробку виняткових ситуацій;
– відображення зрозумілого для користувача повідомлення про помилку;
– контроль формату вхідних даних;
– можливість оновлення та видалення помилкових даних;
– стабільну інфраструктуру системи на базі хмарного сервісу AWS.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для нормальної роботи алгоритмів нормалізації вхідних даних, побудови та навчання моделі необхідні такі технічні засоби:

– графічний процесор, спеціально призначений для роботи з нейронними мережами (наприклад NVIDIA T4);
– центральний процесор достатньо 4-ядерний;
– оперативна пам’ять – 16 Гб;
– SSD-накопичувач на 5 Гб;
– операційна система може бути будь-яким дистрибутивом Linux але повинно бути встановлене ядро для Julia.

Для роботи модулів обслуговування моделі достатньо буде стандартної конфігурації, головне, щоб була встановлена JVM не нижче 11 версії.

Також система працює з NoSQL базою даних DynamoDB, отже вона повинна бути або розгорнута віддалено або локально.

1.5.4. Вимоги до інформаційної та програмної сумісності

Апаратна сумісність модуля роботи моделі зводиться до рівня паралелізму обчислювальної системи, щоб модель могла бути навчена достатньо швидко. Тобто для цього модуля потрібно продуктивне апаратне забезпечення, а саме графічний процесор. Для інших модулів немає апаратного рівня сумісності.

З точки зору програмної сумісності потрібно лише ядро Julia та JVM – без цих встановлених завчасно програмних засобів система не буде працювати.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

Основне функціональне призначення розроблюваної інформаційної системи полягає в організації процесу навчання та тестування моделі машинного навчання DCGAN. В рамках цієї задачі система забезпечує такі можливості:

– зберігання наборів даних для навчання (це графічні дані);

– реалізовано базові операції створення, оновлення, читання та видалення наборів тренувальних даних;

– зберігання станів навченої моделі;

– запускати тренування моделі на вказаному наборі даних;

– тестувати роботу моделі, отримуючи згенеровані зображення.

Для потенційного користувача експлуатаційне призначення системи полягає в створення та тренуванні моделі типу DCGAN для подальшого використання її в своїх системах, маючи при цьому оптимальний для заданої навчальної вибірки масив значень параметрів моделі.

Особливістю системи також є те, що вона розгорнена на базі сервісу хмарних обчислень AWS і має документацію для REST API у вигляді інтерактивного інтерфейсу Swagger.

2.2. Опис застосованих математичних методів

Найбільше математичних методів було застосовано в центральному елементі системи – архітектурі нейронної мережі та алгоритмі її навчання. Далі представлені подробиці цих двох аспектів.

Генеративна змагальна мережа (GAN) – це клас моделей машинного навчання, розроблений Яном Гудфеллоу та його колегами в червні 2014 року. Дві нейронні мережі змагаються один з одним у так званій «грі з нульовою сумою» (гра, у якій гравці мають повністю протилежні цілі, а виграш одного буде

означати програш іншого). Загалом це така архітектура моделі, яка дозволяє оцінити генеративні можливості системи через процес змагання, у якому ми одночасно навчаємо дві підмоделі: генеративну модель G, яка фіксує розподіл даних; і дискримінаційну модель D, яка оцінює ймовірність того, що примірник вхідних даних надійшов з навчального набору, а не відноситься до тих, які були згенеровані моделлю G. Процедура навчання для G полягає в тому, щоб максимізувати ймовірність помилки при розпізнанні фальшивки моделлю D. Ця структура відповідає грі «minimax» для двох гравців. У просторі довільних функцій G і D існує єдине рішення, у якому G відтворює розподіл навчальних даних, а ймовірність помилки D дорівнює всюди 1/2. У випадку, коли G і D є багатошаровими персептронами (повнозв'язними нейронними мережами), всю систему можна навчити за допомогою методу зворотного поширення помилки. Немає потреби в будь-яких ланцюгах Маркова або розгорнутих мережах наближених висновків під час навчання або генерації вибірок. Експерименти демонструють потенціал такої системи шляхом якісної та кількісної оцінки згенерованих зразків [4].

Ян Гудфеллоу з колегами розробили доволі дієву та ефективну модель для вирішення зазначених задач. Розглянемо детально кожен елемент моделі.

Першим розглянемо дискримінатор (в деяких варіаціях GAN, таких як WGAN, його ще називають критиком). Це багатошаровий персептрон, якому на вхід подається масив даних певної розмірності. Розроблена модель працює із зображеннями, тож далі будуть розглянуто різні методи нормалізації зображень для обробки. Вихід дискримінатора – це одне значення в діапазоні від 0 до 1, що відображає вірогідність належності вхідного примірника до тренувальної вибірки (0 – відповідає згенерованим даним, 1 – даним із тренувального набору). У варіації моделі WGAN на виході критика видається числова оцінка від 0 до певного значення, яке може бути необмеженим. Алгоритм навчання при цьому має свої особливості та специфіку.

Генератор – це також багатошаровий персептрон, але має обернену структуру – вхідних нейронів менше ніж вихідних. Вхідний вектор значень – це

так званий шум, який складається із випадкового набору числових значень в діапазоні від -1 до 1. На виході має розмірність таку ж як на вході у дискримінатора. В залежності від форми вихідних даних є декілька способів збільшення розмірності через шари нейронної мережі. Вони будуть розглянуті далі разом із методами нормалізації зображень для обробки моделлю.

Як відомо із теорії штучних нейронних мереж, кожен нейрон на кожному шарі повинен застосовувати функцію активації над виваженою сумою вхідних значень з урахуванням також додаткових відхилень. Ян Гудфеллоу у своїй роботі використовував так званий випрямлений лінійний нейрон (функція активації має назву ReLU) у комбінації із стандартною сигмоїдою для генератора та функцією активації maxout (на виході нейрона береться максимальне значення серед розрахованих виважених входів) для дискримінатора [5]. Проте у даній роботі буде використано функцію активації Leaky ReLU (від'ємні значення не відкидаються повністю, а пригнічуються із певним коефіцієнтом) як для генератора, так і для дискримінатора. Значення коефіцієнта для Leaky ReLU обрано 0.2, виходячи із емпіричних даних, які отримали Алек Редфорд та Люк Метс у свій роботі, присвяченій більш складній та потужній моделі DCGAN [6]. Аргументація використання функції активації Leaky ReLU полягає в тому, що ця функція зменшує прояв проблеми зникання градієнту [7]. На вихідному шарі генератора було використана функція активації гіперболічний тангенс (tanh), тому що значення кожного згенерованого пікселя повинно бути від -1 до 1. На вихідному шарі дискримінатора використовується стандартна сигмоїда, проте для моделі WGAN вихідне значення може не нормуватись взагалі, тоді змінюється функція помилки для правильної роботи алгоритму навчання.

Розглядалось два основних методи нормалізації вхідної матриці даних:

1. Перетворення матриці у вектор та нормалізація значень яскравості пікселів у діапазон від -1 до 1. Цей підхід придатний для невеликих зображень (наприклад із набору MNIST, де кожне зображення має розмір 28 на 28 пікселів). Але для великих матриць вхідних даних, а особливо коли є декілька каналів

(наприклад кольорові зображення RGB), такий спосіб нормалізації вхідних даних неприйнятний, бо це потребуватиме величезні обчислювальні потужності.

2. Використання технології згорткової нейронної мережі (CNN), які є спеціалізованим типом нейронної мережі для обробки даних, яка має топологію сітки. Приклади застосування включають обробку часових рядів, які можна розглядати як одновимірну сітку, що бере вибірки через регулярні інтервали часу, і графічних даних, які можна розглядати як двовимірну сітку пікселів. Згорткові мережі були надзвичайно успішними у практичному застосуванні. Назва «згортка нейронна мережа» вказує на те, що мережа використовує математичну операцію під назвою згортка. Згортка – це спеціалізований вид лінійної операції, в теорії обробки зображень його ще називають локальним оператором або фільтром. Згорткові мережі – це нейронні мережі, які використовують згортку замість загального множення матриці принаймні в одному або декількох зі своїх шарів [8]. Отже, вхідна матриця даних в такій моделі залишається незмінною, може складатися із декількох каналів. Проте для нашої моделі необхідно значення кожного пікселю нормалізувати у діапазон від -1 до 1. Загалом у такої моделі є своя назва – глибока згорткова генеративна змагальна мережа (DCGAN). Для генератора така модель передбачає шари оберненої згортки.

В результаті експериментів і тестів найбільш оптимальним типом нейронної мережі для вирішення поставленої задачі виявився саме DCGAN.

Тепер представимо код на Julia із застосуванням фреймворку Flux, який відображає модель DCGAN :

```
function create_discr(image_size::Integer)
    discr_design = [LayerDesign(4, 2, 1), LayerDesign(4, 2, 1)];
    return Chain(
      Conv((discr_design[1].kernel, discr_design[1].kernel), 3 => 64; stride =
            discr_design[1].stride, pad = discr_design[1].pad),
      x->leakyrelu.(x, 0.2f0),
      x -> customDropout(x, 0.3f0, train_mode),
```

```
        Conv((discr_design[2].kernel, discr_design[2].kernel), 64 => 128; stride =
            discr_design[2].stride, pad = discr_design[2].pad),
            x->leakyrelu.(x, 0.2f0),
            x -> customDropout(x, 0.3f0, train_mode),
            Flux.flatten,
            Dense(culc_size(discr_design, image_size) ^ 2 * 128, 1))
    end
    function create_gen(image_size::Integer)
        gen_design=[LayerDesign(4,2,1),LayerDesign(4,2,1),LayerDesign(5,1,2)];
        size = culc_size(gen_design, image_size);
        return Chain(Dense(latent_dim, size ^ 2 * 256),
            BatchNorm(size ^ 2 * 256, x -> leakyrelu(x, 0.2f0)),
            x -> reshape(x, size, size, 256, :),
            ConvTranspose((gen_design[3].kernel,gen_design[3].kernel),256=>128;
                    stride = gen_design[3].stride, pad = gen_design[3].pad),
            BatchNorm(128, x -> leakyrelu(x, 0.2f0)),
            ConvTranspose((gen_design[2].kernel,gen_design[2].kernel),128=>64;
                    stride = gen_design[2].stride, pad = gen_design[2].pad),
            BatchNorm(64, x -> leakyrelu(x, 0.2f0)),
            ConvTranspose((gen_design[1].kernel,gen_design[1].kernel),
                    64=>3, tanh; stride = gen_design[1].stride,
                    pad = gen_design[1].pad))
    end
```

Далі на рис. 2.1 представлено цю ж моделі у більш наочному вигляді. На схемі також зображено логічний зв’язок між дискримінатором та генератором, а також набір тренувальних примірників.

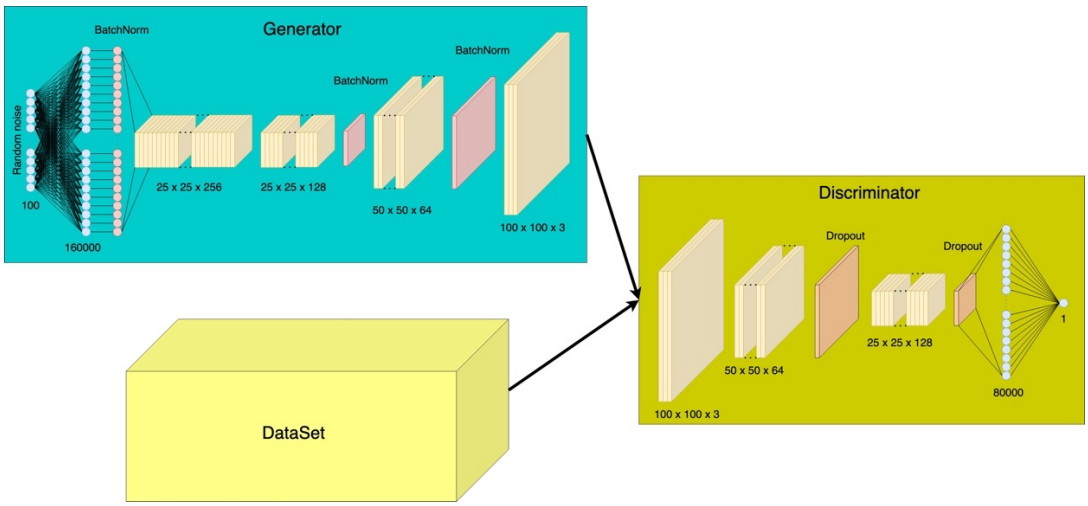

Рис. 2.1. Модель DCGAN

В зазначеному коді та на діаграмі є декілька додаткових елементів, які необхідні для покращення процесу навчання. Серед них BatchNorm та Dropout. Розглянемо їх більш детально.

Існує так звана «пакетна нормалізація» або «нормалізація по вибірці» (Batch Normalization), яку запропонували Сергій Іові та Крістіан Сегеді з компанії Google в 2015 році. Цей метод виконує нормалізацію частини архітектури моделі та робить це для кожного навчального міні-набору. Такий підхід дозволяє ефективно боротися з феноменом «внутрішнього коваріантного зсуву». Пакетна нормалізація дозволяє використовувати набагато вищі темпи навчання і бути менш обережними в ініціалізації гіперпараметрів моделі. Він також діє як регулятор, у деяких випадках зменшуючи необхідність використання методу Dropout (про який буде йтися далі). Суть цього методу полягає в статистичному аналізі та відповідному корегуванні проміжних результатів активацій нейронів у відповідності з нормалізаціє статистичного розподілу цих значень всередині вибірки вхідних примірників. Далі приведене аналітичне представлення цієї ідеї [9].

Математичне очікування в наборі розраховуємо так:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i, \tag{2.1}$$

де μ_B – математичне очікування в міні-наборі B;

m – кількість елементів в міні-наборі;

x_i – значення і-ої активації певного нейрона в міні-наборі B.

Середньоквадратичне відхилення розраховуємо так:

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2, \tag{2.2}$$

де σ_B – середньоквадратичне відхилення в міні-наборі B.

Нормалізоване значення активації нейрона розраховуємо на основі формули (2.1) для математичного очікування та формули (2.2) для середньоквадратичного відхилення так:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}, \tag{2.3}$$

де $\hat{x}_i$ – нормалізоване значення і-ої активації нейрона в міні-наборі B;

ϵ – додаткове коригуюче відхилення.

Масштабоване (за рахунок коефіцієнта γ) та зміщене (за рахунок доданка β) нормалізоване значення розраховуємо на основі формули (2.3) так:

$$y_i = \gamma\hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i), \tag{2.4}$$

де y_i – вихідне значення функції;

γ – коефіцієнт масштабування;

β – коефіцієнт зміщення;

$BN_{\gamma,\beta}(x_i)$ – позначення BatchNorm функції.

Параметри γ та β відносяться до параметрів, які потребують корегування в процесі навчання, тому для них розраховується свої градієнти.

Цей підхід показав доволі високу ефективність в процесі навчання розглянутого типу нейронної мережі, тому саме його було обрано для розробки ефективної конфігурації системи.

Є ще один методи, який також показав певну ефективність для стабілізації процесу навчання мережі. Цей метод більш простий у реалізації, але має один недолік – необхідність генерації випадкового числа, що додає певної невизначеності внутрішньої роботи мережі. Метод називається «відкидання» (Dropout), винайдений Нітішем Срівастава та колегами із університету в Торонто у 2014 році. Цей підхід допомагає вирішити проблему «перенавчання мережі». Ключова ідея полягає в тому, щоб під час навчання випадковим чином відкидати нейрони (разом з їх вагами та зв'язками з іншими нейронами) із нейронної мережі. Це запобігає надмірній коадаптації нейронів. Під час навчання вилучається вибірка з певною кількістю різних «потоншених» мереж. Під час тестування можна відчути ефект усереднення прогнозів усіх цих розріджених мереж, просто використовуючи одну нерозріджену мережу, яка має менші ваги. Це значно зменшує ефект перенавчення та дає значні покращення порівняно з іншими методами регуляції. «Відкидання» покращує продуктивність нейронних мереж в завданнях «навчання з вчителем» [10]. Для написання такого алгоритму, щоб його можна було запускати на базі графічного процесора, було необхідно створити власну реалізацію для Dropout, яка наведена нижче:

```
function customDropout(x, p::Float32, active::Bool)
    if (!active)
        return x
    end
    y = rand!(similar(x, size(x)))
    y = broadcast(el -> el>p ? 1/(1-p) : 0f0, y)
    return x .* y
end
```

Обидва методи працюють тільки в циклі навчання, але повинні бути вимкнені при тестуванні та експлуатації мережі. В результаті експериментів

ефективною виявилась конфігурація коли BatchNorm застосовується для генератора, а Dropout – для дискримінатора. Це обумовлено тим, що дискримінатор не повинен навчатися занадто швидко, а генератор повинен бути більш стабільним в процесі навчання, щоб не випадати в крайнощі.

Другий математичний аспект – алгоритм навчання моделі. Суть процесу навчання полягає у корегування вагових коефіцієнтів, коефіцієнтів зміщень нейронів, коефіцієнтів масштабування γ та зміщення β для шару BatchNorm та інших параметрів у відповідності із типом оцінки якості роботи моделі та в напрямку підвищення цієї якості. Як відомо із теорії нейронних мереж, існує три основних типи навчання нейронних мереж: з вчителем, без вчителя, з підкріпленням. Навчання з вчителем передбачає наявність чіткого розподілення тренувальної вибірки на класи, тобто розподіл даних відомий завчасно. Навчання без вчителя призначене для пошуку кластерів в множині даних, у яких не має чіткого розподілу по класам. Навчання з підкріпленням передбачає оцінку результату роботи нейронної мережі від середовища, яке здатне реагувати на діяльність моделі. У випадку з моделями GAN та DCGAN на загальному рівні застосовується підхід навчання з вчителем. Для такої моделі існує всього два класи: справжні та згенеровані дані. Проте якщо розглядати окремо дискримінатор (критик) та генератор, то для першого це навчання з вчителем, а от для другого – навчання з підкріпленням. Генератор корегує свої налаштування у відповідності із реакцією та оцінкою середовища, яким для ного виступає дискримінатор (який в свою чергу теж навчається, але має при цьому чіткий розподіл тренувальних даних). Отже, модель GAN унікальна по своїй суті, адже вона поєднує два підходи до побудови алгоритму навчання.

Однією із найважливіших частин алгоритму навчання є функція помилки. Їх існує багато та кожна з них ефективна у певній ситуації. Дійсно, від правильності оцінки роботи моделі залежить швидкість та ефективність процесу навчання. У моделі GAN там DCGAN прийнято застосовувати функцію помилки binary cross-entropy або logistic binary cross-entropy. Аналітична аргументація доцільності використання саме цього типу функцій помилки дуже добре виклав

Ян Гудфеллоу у своїй роботі [4]. Тут буде лише зазначено коротке роз'яснення суті бінарної перехресної ентропії.

Для розуміння поняття «ентропія» в теорії інформації треба зазначити, що з точки зору теорії статистики «інформація» еквівалентна невизначеності випадкової величини. Тобто інформація відображає невизначеність результату якоїсь події. Це можна розуміти так, що чим більше інформації в системі, тим більш невизначеним є результат роботи цієї системи. Для визначення кількості цієї інформації, а отже рівня невизначеності системи, було введено поняття «ентропія». Як відомо, ентропія (вже в контексті теорії інформації) – це міра невизначеності системи, міра кількості інформації. Формула ентропії:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log p(x_i), \tag{2.5}$$

де H(X) – ентропія випадкової величини;

$X = \{x_1, x_2, \ldots, x_n\}$ – випадкова величина, яка може мати n станів;

p – вірогідність і-того стану випадкової величини.

В перехресній ентропії, в свою чергу, має місце порівняння двох різних розподілів. Вона відображає відстань від розподілу p до розподілу q. В нашому випадку в якості p виступає розподіл очікуваних вихідних значення моделі, а q – це розподіл дійсних вихідних значень моделі. Ціль навчання моделі – мінімізація цієї відстані. Проте існує два поняття, які відображають відстань між розподілами: відносна ентропія (або ще називають відстань Кульбака-Лейблера) та перехресна ентропія. Різниця полягає в тому, що для відносної ентропії рівність розподілів визначається нульовим значенням цієї функції, а для перехресної ентропії рівність розподілів дорівнює значенню ентропії розподілу p. Для навчання мережі використовується саме перехресна ентропія, бо для відносної ентропії розподіл p є фіксованим і потрібно підганяти саме розподіл q, а в перехресній ентропії немає такого обмеження [11]. Загальна формула перехресної ентропії з n можливих станів випадкової величини X зазначено далі:

$$H(p, q) = -\sum_{i=1}^{n} p(x_i) \log q(x_i), \tag{2.6}$$

де q – другий розподіл випадкової величини X.

Бінарність цієї ентропії обумовлена тим, що в моделі GAN є всього два класи для розпізнання даних: справжні та згенеровані. Тому можливих станів випадкової величини всього два. Формула бінарної перехресної ентропії:

$$H(p, q) = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})] \tag{2.7}$$

де p = {y, 1-y} – очікувані вихідні значення;
q = {ŷ, 1-ŷ} – дійсні вихідні значення.

Для правильного розрахунку функції помилки на наборі даних треба знайти середнє значення цих ентропій. Формула виглядає так:

$$L(w) = \frac{1}{N}\sum_{n=1}^{N} H(p_n, q_n) = -\frac{1}{N}\sum_{n=1}^{N}[y_n \log \hat{y}_n + (1 - y_n) \log(1 - \hat{y}_n)], \tag{2.8}$$

де L(w) – функція помилки, аргументом якої є множина параметрів моделі;
n = {1, 2, …, N} – порядкові номери вхідних примірників.

Логарифмічна бінарна перехресна ентропія – це те ж саме, що бінарна перехресна ентропія, але для покращення якості роботи методу градієнтного спуску над вихідним значенням застосовується функція сигмоїди.

Ціль навчання моделі, як вже зазначалось раніше, є знаходження мінімуму цієї функції. Для цього зазвичай використовується метод градієнтного спуску та його модифікації (але є альтернативи, наприклад, еволюційні алгоритми). Разом

з тим, дискримінатор та генератор мають однаковий принцип по мінімізації функції помилки, проте розподіл очікуваних вихідних значень p для них різний. Для дискримінатора згенерованим примірникам відповідає значення 0 (примірникам із тренувального набору – 1), а для генератора – навпаки, згенерованим примірникам відповідає значення 1. При цьому важливо, щоб в процесі навчання корегування параметрів кожної мережі відбувалось у свій час.

Під час навчання нейронної мережі для корегування кожного параметру (це ваги, зсуви, масштабуючі та коригуючі коефіцієнті для специфічних шарів, таких як BatchNorm) необхідно знайти відповідне значення градієнту і відняти його виважене значення від попереднього значення параметра. Ваговий коефіцієнт градієнту – це, так звана, швидкість навчання, і це один із гіперпараметрів моделі (тобто таких, що підбираються вручну). У цьому полягає сутність принципу градієнтного спуску. Принцип оберненого розповсюдження помилки, в свою чергу, описує безпосередньо методику розрахунку градієнтів кожного параметру. Цей алгоритм був винайдений у 1986 році Девідом Румелхартом, Джефрі Хінтоном та Рональдом Вільямсом, а стаття була оприлюднена найбільш авторитетним науковим журналом Nature [12]. Загальна формула градієнтного спуску для одного параметру, який підлягає коригуванню під час навчання:

$$\hat{w}_j = w_j - r * \frac{\partial L}{\partial w_j}, \tag{2.9}$$

де w_j – j-тий параметр моделі за поточну епоху навчання;

$\hat{w}_j$ – кориговане значення j-тий параметр моделі, яке буде використане для обробки наступної епохи, якщо така буде;

r – коефіцієнт швидкості навчання (підбирається емпіричним шляхом);

L – функція помилки.

Алгоритм розрахунку градієнту параметру не буде розглядатися в аналітичному вигляді в цій роботі, адже він для кожного параметру свій, а таких

параметрів більше двох мільйонів в моделі. За необхідності можна звернутися до оригінальної статті, авторами якої цей метод і був винайдений [12]. Істотних змін алгоритм зворотного розповсюдження помилки не отримав в цій роботі, лише були застосовані різні оптимізатори на рівні формули градієнтного спуску.

Однак цей підхід до корегування параметрів моделі в чистому вигляді доволі повільний і потребує багато обчислень. Також для складних сучасних моделей глибокого навчання цей метод без введення певних корективів виявився взагалі неефективним, а інколи навіть неспроможним досягти оптимального набору значень параметрів. Це було спричинено проблемою занадто швидкого або занадто повільного ковзання по функції помилки в пошуках глобального мінімуму. Також проблему складали локальні мінімуми, з яких цей алгоритм був неспроможний вивести. Основна причина цих проблем – це ручне підбирання гіперпараметру швидкості навчання для кожної конкретної архітектури моделі та кожного типу задачі. Саме тому було винайдено багато різних підходів для оптимізації процесу корегування параметрів, а саме динамічна зміна швидкості навчання. Більше того, сучасні алгоритми оптимізації спроможні навіть розраховувати динамічні коефіцієнти швидкості для кожного конкретного параметру моделі індивідуально. Здебільшого, ці методи використовують статистичні підходи до обчислень та різні поняття з теорії ймовірності.

В багатьох експериментах доволі добре себе показав алгоритм ADAM. Він базується на алгоритмі RMSprop та SGD з моментом. Для кращого розуміння необхідно спочатку визначити основні ідеї RMSprop та SGD з моментом. Почнемо з другого, SGD з моментом (Stochastic Gradient Descent with momentum) – це звичайний алгоритм градієнтного спуску, але замість градієнта на один примірник вхідних даних розраховується момент першого порядку або ще називають рухомим середнім чи математичним очікуванням (застосовується емпірична формула для спрощення розрахунків) над значеннями градієнта для набору даних; при цьому коефіцієнт швидкості навчання залишається сталим. Формула моменту першого порядку для t-ого примірника із навчальної вибірки:

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * \frac{\partial L_t}{\partial w_j}, \quad (2.10)$$

де m – момент першого порядку;

t – номер примірника в міні-наборі;

β_1 – спеціальний коефіцієнт, оптимальне значення якого визначено емпіричним способом і дорівнює 0.9.

Розширивши дещо формулу (2.9) отримаємо формулу SGD з моментом [13]:

$$\widehat{w}_j = w_j - r * m_t \quad (2.11)$$

Оптимізатор RMSprop (Root Mean Squared Propagation), в свою чергу, за основу бере градієнт, але коефіцієнт швидкості навчання адаптується на основі моменту другого порядку або ще називають нецентрованою дисперсією (теж застосовується емпірична формула для розрахунків) над градієнтами для набору примірників. Із назви цього методу зрозуміло, що береться квадратний корінь від моменту другого порядку (це є нецентрованим середньоквадратичним відхиленням), а коефіцієнт швидкості навчання ділиться на отримане значення. Формула моменту другого порядку:

$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * \left(\frac{\partial L_t}{\partial w_j}\right)^2, \quad (2.12)$$

де β_2 – спеціальний коефіцієнт, оптимальне значення якого визначено емпіричним способом і дорівнює 0.999;

v – момент другого порядку.

Розширивши дещо формулу (2.9) отримаємо формулу RMSprop [14]:

$$\widehat{w}_j = w_j - \frac{r}{\sqrt{v_t}} * \frac{\partial L}{\partial w_j} \quad (2.13)$$

Оптимізатор ADAM комбінує обидва підходи: замість градієнту застосовується момент першого порядку, а коефіцієнт швидкості навчання корегується за допомогою ділення на квадратний корінь від моменту другого порядку над градієнтом. Для кращого корегування швидкості навчання особливо на початку, щоб не дати алгоритму занадто швидко ковзати по функції помилки, значення моментів в оригінальній статті (стаття 2015 року за авторством Дієдеріка Кінгма та Джиммі Леі Ба) ще додатково корегуються за допомогою «корекції зміщення» [15]. Формули для корекції зміщення:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \tag{2.14}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2.15}$$

Розширивши дещо формулу (2.9) результатами з формул (2.14) та (2.15) отримаємо кінцеву формулу ADAM [16]:

$$\hat{w}_j = w_j - r * \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \tag{2.16}$$

де ϵ – додатковий коригуючий параметр, значення якого зазвичай не враховується в реальних реалізаціях алгоритму або є малим та сталим задля запобігання помилки обчислення «ділення на нуль».

Більшість з реалізацій цих математичних методів покладені на фреймворк Flux для мови програмування Julia.

2.3. Опис використаних технологій та мов програмування

Для реалізації поставленого завдання було написано код на двох мовах програмування. Модуль, який відповідає за побудову, тестування та навчання моделі DCGAN, написаний на мові програмування функціональної парадигми

Julia. Для реалізації базових математичних методів машинного навчання застосовано фреймворк Flux. Модуль керування машинами SageMaker, на базі яких реалізовано попередній модуль, та модуль керування наборами даних для навчання – обидва написані на мові програмування об'єктно-орієнтованої парадигми Java. Для реалізації методів взаємодії з базою даних, хмарними сервісами AWS та надання REST API для представлення інтерфейсу спілкування з модулями системи було використано фреймворк Spring Boot. В якості сховища даних використано NoSQL базу даних DynamoDB та сховище файлів S3. Далі більш докладно про кожну технологію та мову програмування.

Julia – це високорівнева, високопродуктивна, мова програмування з динамічною типізацією та базується, здебільшого, на функціональній парадигмі. Хоча це мова загального призначення і може використовуватися для написання будь-якого додатка, багато з його функцій добре підходять саме для математичного аналізу та обчислювальної науки.

Відмінні аспекти структури мови Julia включають систему типів з параметричним поліморфізмом та з множинною диспетчеризацією як основною парадигмою в цій мові. Julia підтримує багатопоточні, паралельні та розподілені обчислення (з використанням MPI або OpenMP), а також прямий виклик бібліотек C і Fortran без спеціального коду «склеювання». Julia використовує компілятор на базі технології JIT (just-in-time), проте в контексті мови Julia це скоріше JAOT (just-ahead-of-time), адже Julia компілює весь вихідний код у машинний перед запуском.

Julia має «збірник сміття», використовує оперативне оцінювання та включає ефективні бібліотеки для обчислень з плаваючою комою, лінійної алгебри, генерації випадкових чисел і зіставлення регулярних виразів.

Роботу над Julia розпочали у 2009 році Джефф Безансон, Стефан Карпінскі, Вірал Б. Шах та Алан Едельман, які вирішили створити вільну мову, яка була б високорівневою та достатньо швидкою для обчислень великих та складних математичних моделей. 14 лютого 2012 року команда запустила веб-сайт з дописом у своєму блозі, що пояснює місію мови. Розробники цієї мови мали

досвід роботи з різними мовами програмування, такими як MATLAB, Lisp, R, Python, Ruby, Perl, Java, C та навіть shell. Тому їхньою ціллю було створення такої мови, яка б увібрала в себе всі переваги цих мов і мала найбільш широке застосування. Тож вони хотіли, щоб нова мова була такою ж простою у вивченні, як Python; швидкою, як C; динамічною, як Ruby; мала таку систему макросів, як Lisp; мала такий же знайомий для математиків синтаксис, як MATLAB; була такою ж зручною в роботі зі статистикою, графіками та діаграмами, як R; щоб була така ж потужна система роботи з рядками, як в Perl; а побудова pipe-команд була взята з shell. Принципи розподілених обчислень розробники Julia взяли з Hadoop та Spark. Принципи поліморфізму були запозичені з ООП мов, при чому в Julia немає ООП в чистому вигляді, як в Java або C#. Також вихідний код ядра цієї мови та всіх фреймворків та бібліотек є відкритим та розповсюджується за ліцензіями MIT та GPL v2 [17].

З моменту запуску 2012 року спільнота Julia зросла, і станом на 2020 рік Julia була завантажена користувачами більш ніж 10000 компаній і використовується в більш ніж 1500 університетах із понад 35 мільйонами завантажень станом на січень 2022 року. А вся екосистема Julia складається з понад 11,8 мільйонів рядків коду (включаючи документацію та тести).

Julia залучила деяких високопоставлених користувачів, від інвестиційного менеджера BlackRock, який використовує його для аналітики часових рядів, до британського страховика Aviva, який використовує його для розрахунків ризиків. З 2015 року Федеральний резервний банк Нью-Йорка використовує Julia для створення моделей економіки Сполучених Штатів (включаючи оцінку шоків COVID-19 у 2021 році), відзначаючи, що за допомогою цієї мови була зроблена оцінка моделі приблизно в 10 разів швидше, ніж попередня реалізація на MATLAB. На конференції JuliaCon 2017 Джеффрі Регієр, Кено Фішер та інші розробники мови оголосили, що проект Celeste (він займався вивченням астрономічного каталогу видимого Всесвіту за допомогою масштабованого байєсовського висновку) використовував Julia для досягнення максимальної продуктивності 1,54 петафлопс з використанням 1,3 мільйона потоків на

суперкомп’ютері Cray XC40 (тоді це був шостий найшвидший комп’ютер у світі). Таким чином, Julia приєднується до C, C++ і Fortran як до мов високого рівня, в яких була досягнута потужність обчислення, вимірювана в петафлопсах.

Крім того, Julia почали використовувати в багатьох науково-дослідницьких проектах, де раніше були застосовані інші мови програмування, такі як Python, MATLAB або R, а деякі проекти взагалі переписують існуючий код на мову Julia. Так, ця мова була обрана альянсом Climate Modeling Alliance як єдина мова для впровадження глобальної кліматичної моделі наступного покоління. Цей багатомільйонний проект спрямований на створення кліматичної моделі земного масштабу, яка дасть уявлення про наслідки та проблеми зміни клімату. Julia почали активно впроваджувати в NASA, наприклад для моделювання динаміки розділення космічних кораблів (той самий алгоритм на Julia виявилась у 15000 разів швидшим, ніж на Simulink/MATLAB). Також працюють над вбудованими системами для управління супутником у космосі з використанням Julia для контролю положення.

Згідно з офіційним сайтом, основні особливості мови:

– множинна диспетчеризація: надання можливості визначати поведінку функції в багатьох комбінаціях типів аргументів;

– система динамічних типів;

– продуктивність наближається до мов зі статичним типом, як-от C;

– вбудований менеджер пакетів;

– макроси, схожі на Lisp, та інші засоби метапрограмування;

– викликати функції C безпосередньо: без обгорток або спеціальних API;

– можливість взаємодії з іншими мовами, наприклад Python з PyCall, R з RCall і Java/Scala з JavaCall;

– потужні можливості, подібні до shell, для керування іншими процесами;

– призначений для паралельних і розподілених обчислень;

– співпроцедури: легкі зелені потоки (особливість в тому, що керування ними виконує не ОС, а віртуальна машина);

– типи, що визначаються користувачем, швидкі, компактні, як і вбудовані;

– автоматичне генерування ефективного спеціалізованого коду;

– елегантні та розширювані перетворення для числових та інших типів;

– ефективна підтримка Unicode, включаючи, але не обмежуючись, UTF-8.

Множинна диспетчеризація – поліморфний механізм, який використовується в загальних мовах об’єктно-орієнтованого програмування (ООП), який використовує успадкування. У Julia всі конкретні типи є підтипами абстрактних типів, прямо чи опосередковано підтипами типу Any, який є вершиною ієрархії типів. Конкретні типи самі по собі не можуть бути підтипами так, як в інших мовах; замість цього використовується композиція.

За замовчуванням середовище виконання Julia має бути попередньо встановлена під час запуску наданого користувачем вихідного коду. Крім того, за допомогою PackageCompiler.jl можна створити окремий виконуваний файл, який не потребує вихідного коду Julia. Також Julia має вбудований менеджер пакетів і включає систему реєстру за замовчуванням. Пакети найчастіше поширюються як вихідний код, розміщений на GitHub, хоча також можна використовувати альтернативні варіанти. Пакети також можна встановити як двійкові файли, використовуючи артефакти. Підтримуються федеративні реєстри пакетів, що дозволяє локально додавати реєстри, відмінні від офіційних.

Julia підтримує Unicode з UTF-8, що використовується для рядків і вихідного коду Julia, тобто в якості назв змінних, констант, функцій можна використовувати різні спеціальні символи або символи неанглійського алфавіту. Це дуже зручно для математиків, які можуть писати код, доволі близький до аналітичних формул та записів на папері.

Офіційний дистрибутив Julia включає інтерактивний цикл читання-оцінювання-друку (REPL) командного рядка з історією пошуку та сесіями роботи спеціальними режимами допомоги та оболонки, які можна використовувати для швидкого експериментування та тестування коду. Julia також підтримується Jupyter Notebook, інтерактивним веб-середовищем розробки в стилі REPL, яке може компілювати та запускати код на базі різних ядр, в томи числі й Julia [18].

Flux – це фреймворк для машинного навчання з відкритим вихідним кодом, написаний на Julia. Основний принцип побудови моделей полягає в накладанні шарів простіших моделей і побудови їх у єдиний ланцюг, з якого виділяється набір параметрів, які можна корегувати для навчання моделі. Також має підтримку взаємодії з іншими пакетами Julia. Наприклад, наявна підтримка графічного процесора та обробка на рівні спеціальних масивів CuArrays. Цей фреймворк має велику перевагу, порівняно з іншими фреймворками машинного навчання такими як TensorFlow, саме в середовищі Julia. Вона полягає в тому, що Flux повністю написаний, адаптований та оптимізований під мову Julia та її особливості. TensorFlow працює значно повільніше ніж Flux, бо його ядро реалізовано на інших мовах програмування, серед яких повільний Python.

Flux підтримує рекурентні та згорткові нейронні мережі. Він також здатний до диференційного програмування через свій пакет Zygote. Flux використовувався також як каркас для побудови нейронних мереж, які працюють з зашифрованими даними, ніколи не розшифровуючи їх. Цей фреймворк в майбутньому може стати базовим для реалізації API з високим рівнем захищеності даних, які використовують моделі машинного навчання. Flux також може ефективно працювати із обладнанням CUDA, графічними процесорами від компанії Nvidia, які мають високу ефективність та швидкодію в задачах машинного навчання [19].

Отже, Julia – це популярна мова в сфері машинного навчання, а Flux – це популярний фреймворк, який реалізує математику машинного навчання.

Базовою особливістю модулів керування ядром моделі GAN та керування наборами даних для навчання є те, що вони мають ознаки серверних додатків, які обслуговують інтерфейс взаємодії з базою даних, іншими зовнішніми системами та надають підтримку багатьох паралельних запитів. Тож необхідно використати найбільш ефективну мову програмування для цих цілей. І найпопулярнішою мовою в цьому сегменті є Java. Ця мова дозволяє побудувати дуже надійний серверний додаток. Далі розглянемо основні переваги Java в цьому напрямку:

1. Статична типізація – прийом, при якому змінна, параметр підпрограми, значення, що повертають функції, пов’язується з типом в момент оголошення і тип не може бути змінений пізніше. Ця особливість дозволяє знайти якісь помилки в коді ще на етапі компіляції (інколи кажуть, що Java-код складно скомпілювати з першого разу, адже будь-які невідповідності та неоднозначності в коді компілятор одразу знаходить). А одна із причин, чому саме мови зі статичною типізацією дуже часто використовуються для побудови систем автоматизації бізнесу, – це висока швидкодія додатку саме для великих систем, в порівнянні з кодом, написаним на мовах з динамічною типізацією, того ж об’єму системи (динамічна типізація у великих системах дуже гальмує її роботу).

2. JVM та збірник сміття – технології, які були створені ще за часів мови SmallTalk, набули великої популярності саме завдяки їх реалізації в Java. JVM (Java Virtual Machine) – набір комп’ютерних програм та структур даних, що використовують модель віртуальної машини для виконання інших програм чи скриптів. JVM використовує байт-код Java, який, як правило, але не завжди, генерується з вихідних кодів мови програмування Java; віртуальну машину також застосовують для виконання коду, згенерованого з інших мов програмування, таких як Scala або Kotlin. JVM доступна для всіх основних сучасних платформ, тому про програми, що скомпільовані у Java байт-код, можна сказати: «написано один раз, працює скрізь». Тобто це і є принцип кросплатформовості і серверний код можна буде розгорнути на різних апаратних та програмних системах. JVM також принесла дуже корисну технологію в Java – збірка сміття (garbage collection). Це один із видів автоматизованого керування пам’яттю. Збирач сміття (garbage collector) – це спеціальний процес програми, який періодично очищує пам’ять від об’єктів, які вже не використовуються. В таких мовах, як C та C++ за очищенням пам’яті програміст повинен слідкувати самостійно, що додає незручностей при розробці, особливо великих систем, призводить до частих помилок, які інколи можуть бути навіть фатальними, також такий код набагато складніший в розумінні та підтримці. Отже, ідея автоматизувати процес керування пам’яттю – дуже слушна. До речі, в мові Julia

також є свій збирач сміття, тож ця технологія має надзвичайно широке застосування в сучасних мовах програмування.

3. JIT-компіляція. У мовах, які працюються на базі віртуальних машин, є значний недолік – це повільність роботи, бо паралельно з основною програмою ще працює віртуальна машини та додаткові системи, такі як збирач сміття. Для подолання цього недоліку була розроблена JIT-компіляція (компіляція на льоту) – технологія збільшення продуктивності систем, що виконують програмний код, шляхом трансляції байт-коду в машинний код безпосередньо під час роботи програми. Алгоритм цього типу компіляції здатний зробити виконуваний машинний код набагато швидшим за рахунок оптимізації деяких ділянок основного коду програми. Це дає певну переваги для мови Java, адже великі навантажені системи повинні бути швидкими, а найкращій спосіб це зробити – виконувати оптимізацію на ходу, виходячи із статистки роботи вже виконаного коду. Програмісту, як людині, дуже складно зробити оптимізацію одразу на етапі написання коду, а якщо цей код дуже великий, то це майже неможливо. Як вже зазначалося, в мові Julia цей принцип дещо видозмінений на користь принципу JAOT, але загалом ідеї схожі.

4. Об’єктно-орієнтоване програмування (ООП) – парадигма програмування, яка каже про те, що все в програмі – об’єкти, які взаємодіють між собою. Очевидно, що такий підхід дуже природній для сприйняття людиною. А отже, програмісту такий підхід до розробки буде набагато простіший. У великих системах оперувати об’єктами набагато зручніше, ніж процедурами або функціями. Також такий код дуже легко підтримувати, а розробка в команді стає доволі зручною. З технічної точки зору ООП – це технологія, яка надає більшої гнучкості мовам зі статичною типізацією, додається можливість динамічного виконання коду, яка притаманна мовам з динамічною типізацією. Отже, ООП позбавляє недоліків мови зі статичною типізацією, при цьому залишаючи всі переваги такого підходу.

Серед мов подібних Java є дуже перспективна та прогресивна мова – C#. Але у цієї мови є один великий недолік – малий вибір фреймворків, бібліотек та

додаткових програмних засобів. А причина цього – монополія корпорації Microsoft на всі продукти, пов’язані з платформою .NET. Java, в свою чергу, підтримується та розробляється спільнотою програмістів з усього світу, а вибір різних фреймворків та додаткових програмних засобів достатньо великий. З цього також випливає той факт, що база знань по Java надзвичайно велика.

Разом з цією мовою часто говорять про її фреймворк Spring. Він робить розробку серверних систем ще більш зручним за рахунок застосування деяких шаблонів проектування, зокрема «інверсія керування» та «ін’єкція залежностей». Основні особливості Spring можуть бути використані будь-яким додатком Java, але є розширення для створення веб-додатків на платформі Java EE. Незважаючи на це, Spring не нав’язує якоїсь конкретної моделі програмування, Spring став популярним в спільноті Java як альтернатива, або навіть доповнення моделі Enterprise JavaBean (EJB).

Один з основних принципів ООП полягає в тому, що класи повинні мати мінімальну взаємозалежність, тобто низьку зв’язність. Інверсія керування (IoC) – це принцип проектування, який дозволяє класам бути слабо зв’язаними і, отже, їх можна легше тестувати й підтримувати. IoC базується на передачі управління об’єктами та їхніми зв’язками контейнерним системами або спеціальним фреймворкам, таким як Spring. IoC – це скоріше принцип, а не шаблон проектування, тож деталі реалізації залежать від конкретної ситуації.

Ін’єкція залежності – це техніка, яка дозволяє відокремлювати об’єкти від їх залежностей. Припустимо, що об’єкту А потрібен метод об’єкта В для реалізації його функціональності. Ін’єкція залежностей передбачає, що замість створення екземпляра класу В у класі А за допомогою оператора new, об’єкт класу В слід ввести в клас А за допомогою одного з наступних методів: ін’єкція конструктором, ін’єкція методом встановлення значення поля класу, ін’єкція інтерфейсом. Докладно ці методи розглядатися не будуть, адже їх реалізація покладена на фреймворк Spring і не була ніяк модифікована [20].

Замість базового фреймворку Spring, в розробленій системі використано Spring Boot. Його особливість полягає в автоматизованій системі

конфігурування. З часом було створено багато стандартних реалізацій різного функціоналу, тому для пришвидшення та спрощення подальшої розробки серверних систем було розроблено так звані стартери Spring Boot. Це дозволяє ще менше приділяти увагу інфраструктурним елементам системи і більше зосередитись на розробці бізнес-логіки додатку. Зокрема, були застосовані стартери для взаємодією з базою даних, побудови REST API і таке інше.

Amazon DynamoDB – це повністю керована запатентована компанією Amazon система керування базами даних на основі технології NoSQL, яка підтримує структури даних типу «ключ-значення», набори яких складають документ (аналог таблиці в реляційних БД), є частиною сервісу хмарних обчислень Amazon Web Services (AWS). DynamoDB надає подібну модель даних як в однойменному сховищі даних Dynamo, але має іншу базову реалізацію. Dynamo мала багаторівневу конструкцію, яка вимагала від клієнта вирішувати конфлікти версій, а DynamoDB використовує синхронну реплікацію в кількох центрах обробки даних для високої довговічності та доступності. Основна причина використання саме цієї БД – простота і лаконічність інтерфейсу взаємодії, а також її повна інтегрованість в AWS, як основної системи NoSQL в цьому хмарному сервісі. Особливості реляційної моделі та її рівні нормалізації є зайвими для розроблюваної системи, тому оптимальним є саме NoSQL підхід.

Amazon S3 або Amazon Simple Storage Service – це служба, яка забезпечує зберігання об’єктів через інтерфейс веб-служби. Amazon S3 використовує ту ж масштабовану інфраструктуру зберігання, що й сам Amazon для роботи своєї мережі електронної комерції. Amazon S3 може зберігати будь-який тип об’єктів, що дозволяє використовувати його як, наприклад, сховище для Інтернет-додатків, резервних копій даних або налаштувань системи, архівних даних, також для аварійного відновлення системи. Часто там зберігають дані для аналітики, як зокрема у випадку з розроблюваною системою – в AWS S3 зберігаються файли наборів даних, станів моделі та згенеровані зображення. Всі ці дані неможливо зберігати в DynamoDB через великий розмір одного такого елементу, тому в БД зберігаються лише деякі посилання на об’єкти в AWS S3.

2.4. Опис структури системи та алгоритмів її функціонування

Розроблювана інформаційна система складається з трьох модулів:

– модуль ядра моделі DCGAN (він реалізує побудову, алгоритм навчання та тестування нейронної мережі);

– модуль керування станами моделі (реалізує зберігання станів моделі, а також запускає, зупиняє модуль ядра DCGAN і надає йому всю необхідну інформацію для тренування та тестування моделі);

– модуль керування наборами тренувальних даних (здійснює верифікацію наборів даних при завантаженні, обробляє запити на створення, отримання, видалення та оновлення тренувальних наборів).

Також наявні два типи сховища даних: DynamoDB та AWS S3. Для наочного представлення структури та деяких алгоритмів розроблюваної інформаційної системи були створені кілька UML діаграм, зокрема загальну діаграму компонентів та дві діаграми послідовностей для відображення алгоритмів тестування та тренування моделі GAN.

На діаграмі компонентів показано, як різні компоненти з’єднані між собою, що утворює більші компоненти або програмні системи чи модулі. Вони використовуються для ілюстрації структури систем будь-якої складності. Діаграма компонентів дозволяє перевірити, що необхідні функціональні елементі системи можуть мати зазначені зв’язки з іншими модулями для реалізації поставленої задачі. Програмісти та розробники використовують діаграми, щоб формалізувати спосіб впровадження певного функціоналу, що дозволить краще приймати рішення щодо розділення роботи між розробниками для ефективної роботи в команді.

Діаграма компонентів розширює інформацію, наведену в елементі позначення компонента. Один із способів ілюстрації наданих і необхідних інтерфейсів взаємодії зазначеним компонентом – це маленький прямокутник, приєднаний до компонентного елемента, його називають «портом» компонента. Також кружечки – це інтерфейси. Якщо такий кружечок цілий, то це означає, що

той компонент, до якого він відноситься, надає цей інтерфейс. А якщо кружечок розірваний, то це означає, що компонент-володар такого кружечка використовує інтерфейс, наданий іншим компонентом. Залежність від компонента до інтерфейсу ілюструється суцільною лінією до компонента. Всередині компонента можуть бути внутрішні компоненти, тобто логічні елементі, які складають та забезпечують певний функціонал цілому компоненту. Також можна використовувати елементи документа із зазначеними полями та їх типами, якщо це, наприклад, таблиця в БД. Внутрішні елементи компонента можуть мати направлений зв'язок між собою, що показує, наприклад, передачу даних в певному напрямку.

Діаграма послідовності (sequence diagram) показує взаємодії процесів, організованих у часовій послідовності в розроблюваній інформаційній системі. Він відображає залучені процеси та послідовність повідомлень, якими обмінюються процеси або елементи системи, необхідні для забезпечення певного функціоналу. Діаграми послідовностей іноді називають діаграмами подій або сценаріями подій. Діаграма акцентує увагу на подіях, які генерують або компоненти системи, або зовнішні фактори (наприклад користувачі). Для деяких складних сценаріїв слід створити діаграму послідовностей для кращого розуміння послідовності операцій в системі для обробки сценарію.

Діаграма послідовності показує у вигляді паралельних вертикальних ліній (лінії життєвого циклу) різні процеси або об'єкти, які живуть одночасно, а у вигляді горизонтальних стрілок повідомлення, якими вони обмінюються, у тому порядку, в якому вони повинні відбуватися для коректної роботи алгоритму. Діаграма послідовності повинна показувати акторів (елементи системи або зовнішні користувачі); повідомлення (методи), викликані цими акторами; повернення значень (якщо такі є), пов'язані з попередніми повідомленнями; позначення будь-яких циклів або області ітерації.

Далі будуть представлені описані типи діаграм з розширеним поясненням.

На рис. 2.2 представлено розроблену діаграму компонентів.

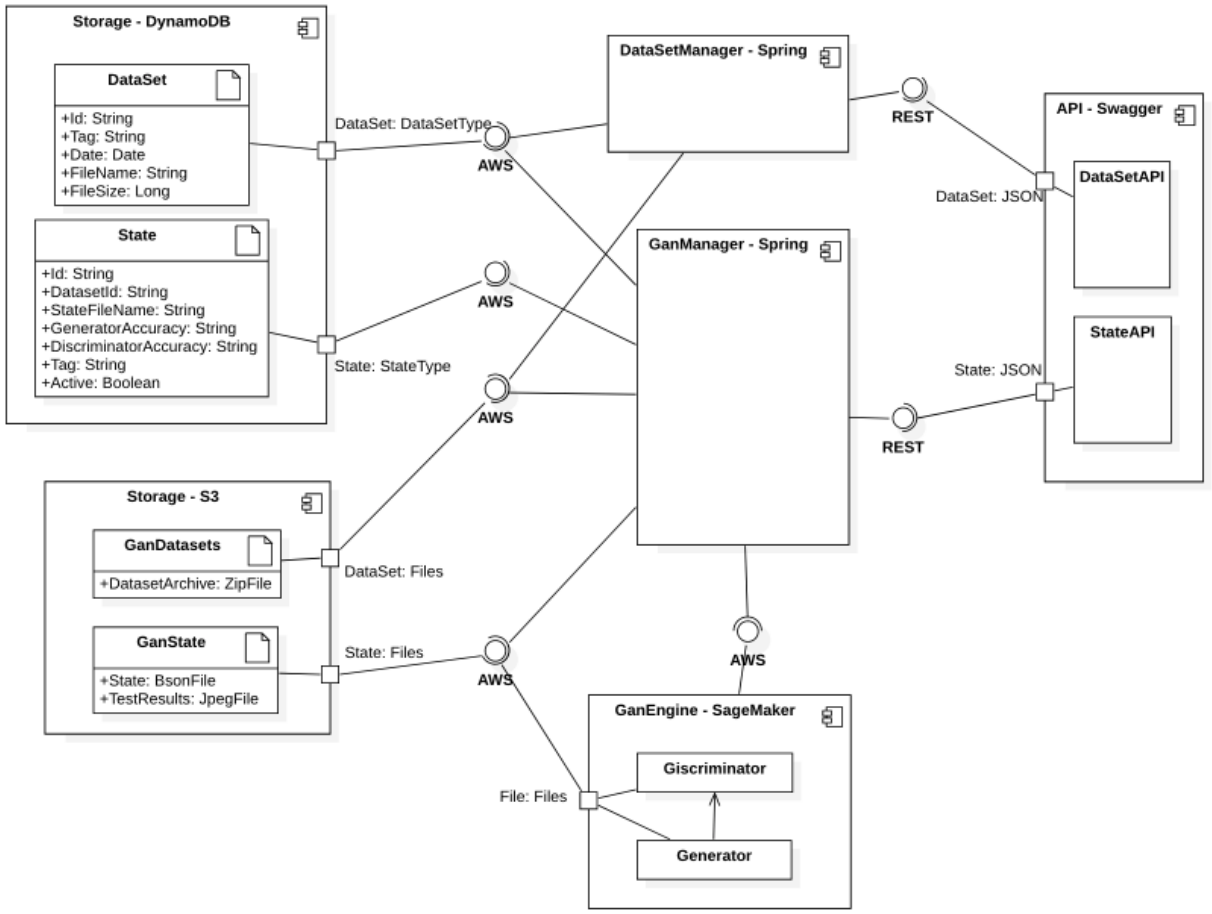

Рис. 2.2. Діаграма компонентів

На цій діаграмі представлені три основні фізичні модулі, два компонента сховища даних, один логічний компонент, який відображає зовнішній інтерфейс взаємодії з інформаційною системою. Назва компонентів складається з двох частин: назва функції, яку виконує компонент; технологія, на якій цей компонент реалізовано. Також зазначені типи портів та інтерфейсів.

Почнемо зі сховища даних. До речі, тут достатньо наочно показано модель даних, поля та їх типи. В БД наявні лише дві моделі, між якими немає чіткого зв’язку, тому немає сенсу розробляти окрему ER-діаграму, адже всі необхідні пояснення подано на діаграмі компонентів. Сховище DynamoDB призначене для метаданих об’єктів, а AWS S3 – для файлів корисних для роботи моделі даних. Також всі сховища лише надають свої інтерфейси та ніколи не роблять самостійних запитів до будь-яких інших компонентів системи.

Документ DynamoDB під назвою DataSet має такі поля:

– id – ідентифікатор набору тренувальних даних, який генерується автоматично і є стандартним UUID;

– tag – будь-який параметр моделі, за яким можна об’єднувати різні набори в один єдиний, у примірників якого є щось спільне (наприклад автор, сюжет або жанр), за допомогою цього поля потенційно можна створювати нові тренувальні набори на основі вже наявних в базі (але цей прецедент не буде розглядатися у цій роботі, адже потребує великої бази наборів для демонстрації);

– date – дата створення тренувального набору (вказується користувачем);

– filename – назва файлу (з урахуванням розширення), який відповідає цьому запису в сховищі GanDatasets;

– fileSize – розмір завантаженого файлу тренувального набору.

Документ DynamoDB під назвою State має такі поля:

– id – ідентифікатор стану моделі, який генерується автоматично і є стандартним UUID;

– dataSetId – ідентифікатор набору тренувальних даних, який передається при запиті на тренування або створення моделі;

– stateFilename – назва файлу (з урахуванням розширення), в якому зберігається стан моделі в сховищі GanState;

– generatorAccuracy – значення функції помилки для генератора;

– discriminatorAccuracy – значення функції помилки для дискримінатора;

– tag – має те ж значення, що й для моделі тренувального набору;

– active – флаг, який показує, чи активна зараз модель (це може бути стан або тренування, або тестування моделі).

AWS S3 складається із «кошиків», в яких можуть бути файли різного типу, а також папки – принцип схожий на файлову систему в ОС. У сховищі AWS S3 зберігають файли, назва яких збігається із ідентифікатором відповідного запису в DynamoDB, а тип (тобто розширення) файлу зазначено на діаграмі. Зокрема, в «кошику» GanDatasets зберігаються лише файли наборів даних у форматі zip-архіву. В GanState зберігаються файли стану в бінарному форматі bson, а також згенеровані зображення у форматі jpg. На додачу, файли зображень зберігаються

всередині папки із назвою, яка відповідає ідентифікатору запису, а самі зображення мають назву, яка є порядковим номером всередині цієї папки. Модуль керування станом моделі збирає файли з папки і формує архів, який віддає користувачу після тренування або тестування моделі. В GanState можуть зберігатися також деякі тимчасові файли, які потрібні для забезпечення процесу тестування та навчання моделі, але про це докладніше буде зазначено далі.

Щодо функціональних модулів, кожен з них має зв'язки з тим чи іншим сховищем через відповідний AWS SDK: в Java за це відповідають спеціальні бібліотеки для роботи з DynamoDB та AWS S3; а в Julia є відповідні пакети.

Код GanEngine представлено в додатку А.1.

Як бачимо з діаграми, модуль GanEngine має тип реалізації на базі машини SageMaker і неписаний на Julia, що вже зазначалося в попередніх розділах. Цей модуль має зв'язок зі сховищем GanState на базі AWS S3, для цього інтерфейсу модуль GanEngine виступає споживачем. Також цей модуль надає інтерфейс через AWS SDK для інших компонентів. Застосовані операції над машиною SageMaker подані в таблиці 2.1.

Таблиця 2.1

Операції на машиною SageMaker

Елемент сервісу	Операція	Опис
Notebook instance	listNotebookInstances	Повертає список наявних машин, які працюють на базі Jupyter Notebook, можна отримати деякі дані про машину, наприклад, статус
	startNotebookInstance	Запускає машину на базі Jupyter Notebook, при цьому виконується завантажувальний скрипт в lifecycle config
	stopNotebookInstance	Зупиняє машину на базі Jupyter Notebook

Продовж. табл. 2.1

Елемент сервісу	Операція	Опис
Lifecycle config	listNotebookInstanceLifecycleConfigs	Повертає список конфігурацій життєвого циклу машини
	updateNotebookInstanceLifecycleConfig	Оновлює скрипти на створення або на запуск машини

Всі ці операції виконує модуль GanManager, який таким чином реалізує три основні дій над цим модулем:

– активація для тренування;

– активація для тестування;

– деактивація машини (призводить до її вимкнення для економії витрат).

Перед кожною операцією виконується перевірка поточного статусу машини, якщо він не прийнятний, то повертається повідомлення про помилку. При активації виконуються певні маніпуляції зі скриптом на запуск машини, який зазначений в конфігурації життєвого циклу. Основною відмінністю з точки зору модуля ядра моделі між активацією на тренування та активацією на тестування є одна команда в скрипті на запуск. Нижче приведений скрипт для активації на тренування:

```
#!/bin/bash
set -e
sudo -u ec2-user -i <<EOF
echo ". /home/ec2-user/anaconda3/etc/profile.d/conda.sh" >> ~/.bashrc
conda run --prefix ~/SageMaker/envs/julia/ julia --eval 'using IJulia;
    IJulia.installkernel("Julia")'
nohup /home/ec2-user/SageMaker/envs/julia/bin/julia /home/ec2-user/
    SageMaker/train-gan.jl &
EOF
```

Далі зазначено скрипт для активації на тестування:

```
#!/bin/bash
set -e
sudo -u ec2-user -i <<EOF
echo ". /home/ec2-user/anaconda3/etc/profile.d/conda.sh" >> ~/.bashrc
conda run --prefix ~/SageMaker/envs/julia/ julia --eval 'using IJulia;
    IJulia.installkernel("Julia")'
nohup /home/ec2-user/SageMaker/envs/julia/bin/julia /home/ec2-user/
    SageMaker/test-gan.jl & EOF
```

Як бачимо відмінність полягає лише в тому, який Julia файл буде запущено. Алгоритм тестування складається з таких кроків:

1. Перевіряємо наявність файлу dcgan.bson в сховищі GanState, якщо його немає, то завершуємо виконання скрипту.
2. Завантажуємо файл стану моделі dcgan.bson.
3. Запускаємо генератор на 9 випадково згенерованих вхідних значеннях.
4. З отриманого результату формуємо колаж згенерованих зображень та зберігаємо його в сховище GanState під назвою dcgan.jpg.

Алгоритм тренування складається з таких кроків (загальний вигляд):

1. Перевіряємо наявність файлу dcgan.zip (тренувальний набір) в сховищі GanState, якщо його немає, то завершуємо виконання скрипту.
2. Завантажуємо файл тренувального набору dcgan.zip та розпаковуємо його, формуючи масив зображень, розбитий на вибірки.
3. Якщо поточний стан моделі присутній (перевіряється наявність файлу dcgan.bson в сховищі GanState), то завантажуємо цей стан, якщо ж немає, то будуємо нову модель.
4. Проводимо зазначену кількість епох навчання.
5. Зберігаємо оновлений стан моделі в файл dcgan.bson в GanState.
6. Формуємо колаж з 9 згенерованих зображень та зберігаємо його в сховище GanState під назвою dcgan.jpg.

В обидвох випадках ознакою завершення роботи скрипту є поява файлу dcgan.jpg в сховищі GanState. Це в свою чергу запускає процес деактивації ядра

моделі. Модуль GanManager проводить перевірку поточного стану модуля ядра моделі кожну хвилину і деактивує його, якщо це можливо. Це необхідно для економії витрат на ресурс SageMaker (його ціна зазначена в наступних розділах).

Модуль керування станом моделі має зв’язки майже з усіма інтерфейсами. При чому надає він тільки один інтерфейс – це REST API, документоване за допомогою Swagger. Цей модуль виконує такі операції над станами моделі:

– запуск тестування стану моделі по заданому ідентифікатору;

– запуск тренування моделі з подальшим формуванням нового стану моделі, це, по суті, і є запит на створення моделі (в якості параметра передається ідентифікатор тренувального набору та кількість епох тренування);

– продовження тренування на основі існуючого стану моделі (в якості параметрів передається ідентифікатор стану та кількість епох);

– отримання згенерованих зображень, беруться всі зображення з відповідної папки в сховищі GanState та формується zip-архів;

– отримання списку наявних станів моделі (список їх ідентифікаторів);

– видалення стану по ідентифікатору.

Докладно розглянемо перші чотири операції. Алгоритм тестування з точки зору модуля керування станом моделі складається з таких кроків:

1. Перевірка активного стану, якщо виявлено, що в даний момент є активний стан, то повертається повідомлення про помилку, адже одночасне тестування на даному етапі розробки системи не підтримується.
2. По ідентифікатору дістається запис з документа DynamoDB State.
3. Копіюється відповідний файл стану моделі з назвою dcgan.bson.
4. Поле-флаг «active» в записі стану змінює своє значення на активне.
5. Завантажується відповідний стартовий скрипт для тестування через конфігурацію життєвого циклу.
6. Відправляється запит в SageMaker на запуск відповідної машини.

Код запуску тестування представлено в додатку А.3.

Для зручності розуміння алгоритму тестування моделі була побудована діаграма послідовностей. Вона наведена на рис. 2.3.

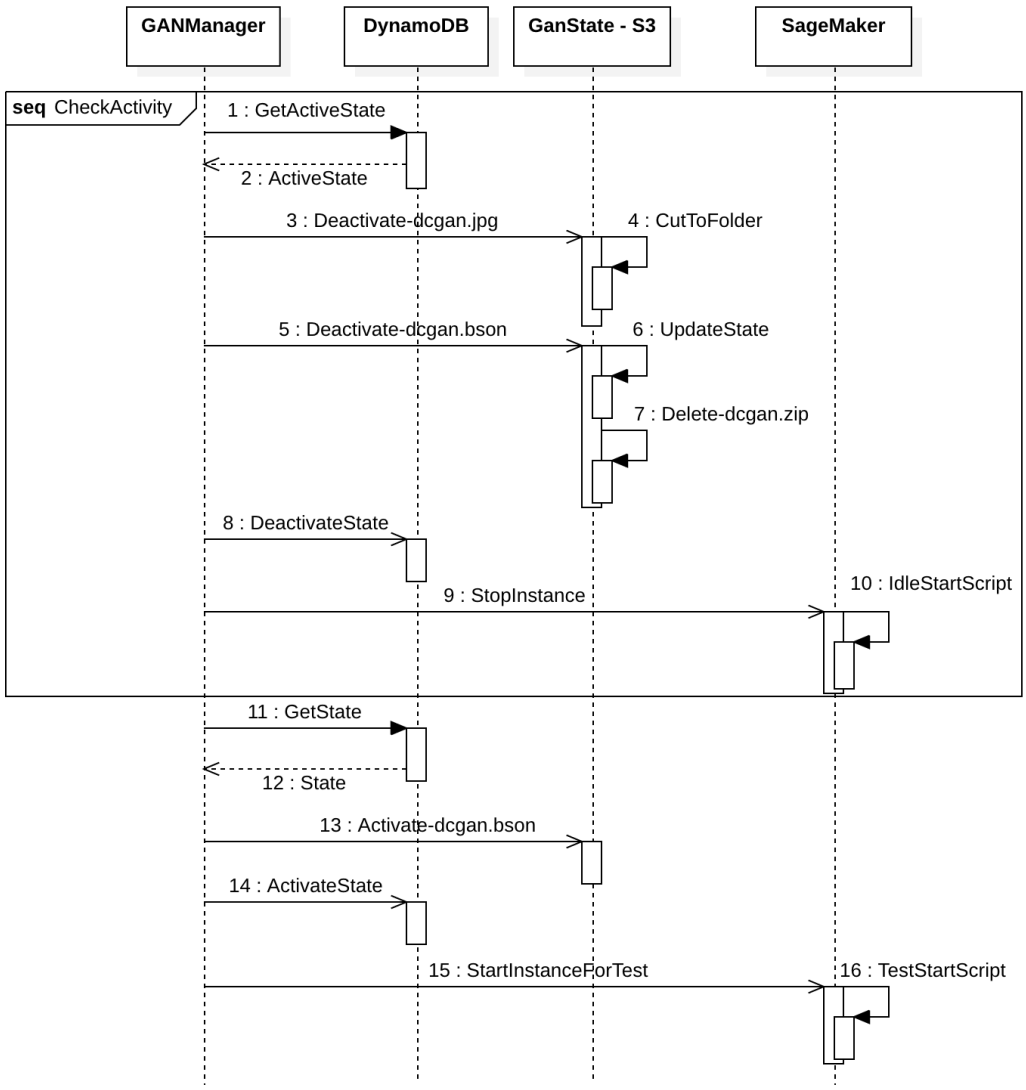

Рис. 2.3. Діаграма послідовностей для алгоритму тестування

Алгоритм тренування з точки зору GanManager складається з таких кроків:

1. Перевірка активного стану, якщо виявлено, що в даний момент є активний стан, то повертається повідомлення про помилку, адже одночасне тренування на даному етапі розробки системи не підтримується.

2. Якщо оброблюваний запит передбачає продовження навчання моделі, то по ідентифікатору відповідний запис в сховищі State дістається, а якщо в запиті передбачено створення нового стану, то формується новий запис в сховищі State.

3. По значенню поля «datasetId» в отриманому записі зі сховища AWS S3 GanDatasets дістається файл навчального набору.

4. Потім цей файл завантажується в сховище GanState з назвою dcgan.zip.

5. Якщо запит передбачає продовження навчання моделі, то відповідний файл стану копіюється з назвою dcgan.bson.

6. Поле-флаг «active» в записі стану змінює своє значення на активне.

7. Завантажується відповідний стартовий скрипт для тренування через конфігурацію життєвого циклу.

8. Відправляється запит в SageMaker на запуск відповідної машини.

Код запуску тренування представлено в додатку А.3.

Для зручності розуміння алгоритму тренування моделі була побудована діаграма послідовностей. Вона наведена на рис. 2.4.

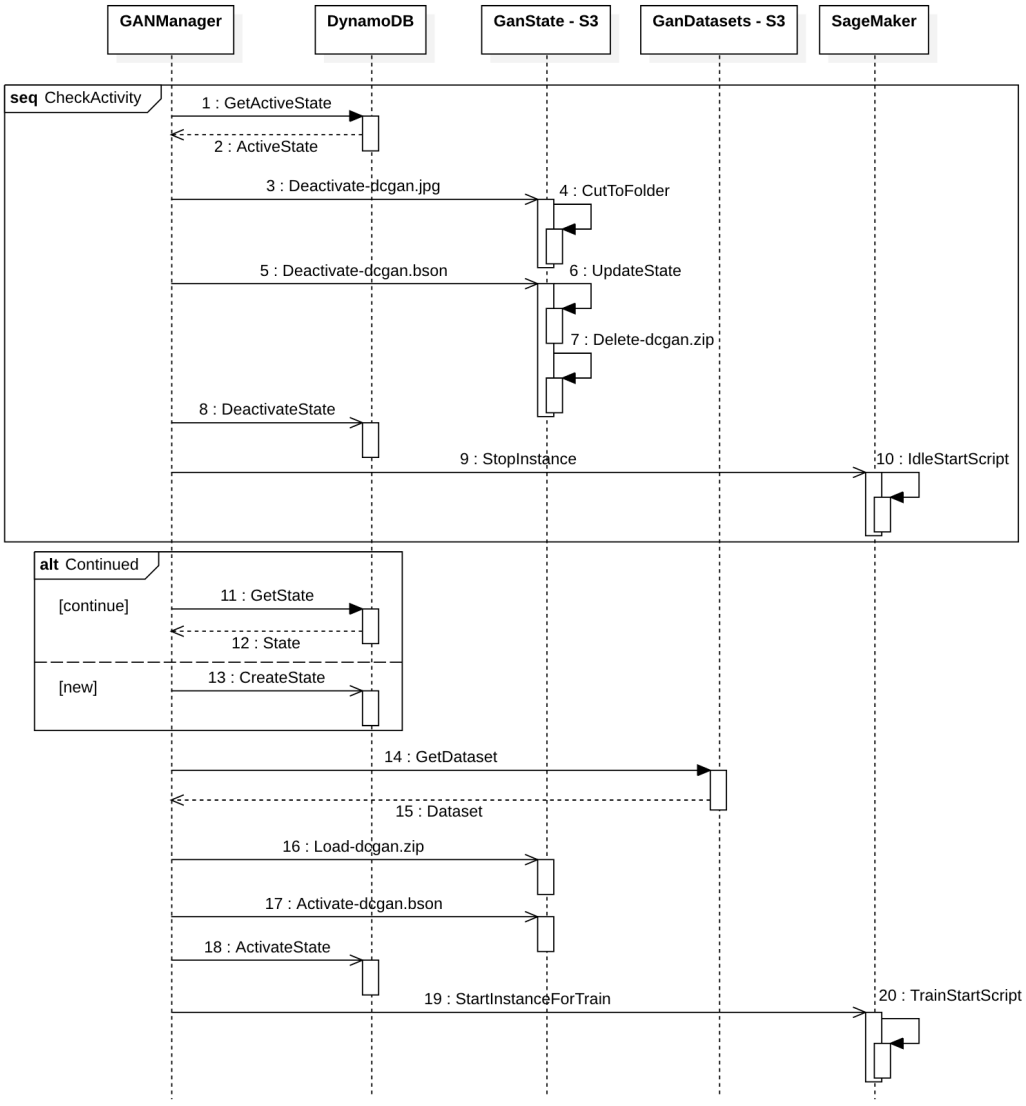

Рис. 2.4. Діаграма послідовностей для алгоритму тренування

Активний стан моделі – це набір ознак, які свідчать про виконання певної обробки ядром моделі. Тож був розроблений певний алгоритм перевірки активності модуля ядра, який виконується кожну хвилину автоматично:

1. Визначення за метаданими моделі, які зберігаються в DynamoDB документі State (в нього є поле-флаг «active», за яким робиться фільтрація), ідентифікатор активного стану, якщо такого не знайшлось, то ядро моделі вважається вільним.

2. По зазначеному ідентифікатору робиться деактивація моделі, яка полягає в певних маніпуляціях з файлами в сховищі GanState, зокрема спочатку визначається наявність маркера завершення роботи ядра моделі – це присутність файлу dcgan.jpg в сховищі GanState, якщо його немає, то деактивація виявилась безуспішною і стан визнається активним.

3. Якщо файл-маркер був знайдений, то він перейменовується та копіюється у відповідну папку, а назва файлу встановлюється відповідною його порядковому номеру в рамках цієї папки.

4. Файл стану моделі оновлюється вмістом файлу dcgan.bson, останній після цього видаляється.

5. Видаляється файл тренувального набору dcgan.zip, якщо він є.

6. Поле-флаг «active» в записі стану змінюється на неактивне.

7. Виконується запит до SageMaker на зупинку відповідної машини.

8. Оновлюється скрипт запуску машини в конфігурації життєвого циклу на нейтральний, без запуску будь-яких скриптів (це зроблено для того, щоб розробник міг вручну запустити машину без прихованого виклику скриптів).

Модуль керування наборами тренувальних даних має зв’язки лише з документом DynamoDB DataSet, з AWS S3 «кошиком» GanDatasets та надає REST API, документоване за допомогою Swagger. Цей модуль виконує базові операції над об’єктами наборів тренувальних даних:

– створення набору, виконується у два етапи: завантаження метаданих про набір, завантаження файлу набору (йому надається відповідна назва та завантажується в сховище GanDatasets);

– читання набору по ідентифікатору, можна окремо завантажити метадані або файл набору;

– отримання списку наявних наборів (список їх ідентифікаторів);

– видалення набору по ідентифікатору.

Код цього модуля подано в додатку А.2.

З точки зору логічного компонента API Swagger є два базові хоста, певні порти яких слухає відповідний модуль. В таблиці 2.2 представлені методи API.

Таблиця 2.2

Методи API

API	Ресурс	Метод	Параметри
DatasetAPI (/dataset)	/getAll	GET	Немає
	/getMetadata/{id}	GET	Ідентифікатор зазначається в path запиту
	/getPictureFile/{id}	GET	Ідентифікатор зазначається в path запиту
	/saveMetadata	POST	Метадані вказуються в тілі запиту у форматі JSON
	/savePicture/{id}	PUT	Ідентифікатор зазначається в path запиту, а також додається файл
	/delete/{id}	DELETE	Ідентифікатор зазначається в path запиту
StateAPI (/gan)	/getAll	GET	Немає
	/getGenerated/{id}	GET	Ідентифікатор зазначається в path запиту
	/test/{id}	POST	Ідентифікатор зазначається в path запиту
	/train/{datasetId}/{epochs}	POST	Ідентифікатор тренувального набору та кількість епох зазначаються в path запиту
	/train/continued/{stateId} /{epochs}	PUT	Ідентифікатор стану моделі та кількість епох зазначаються в path запиту
	/delete/{id}	DELETE	Ідентифікатор зазначається в path запиту

2.5. Обґрунтування й організація вхідних та вихідних даних програми

Вхідними даними всієї інформаційної системи є набори навчальних даних. Тип цих даних – кольорові або чорно-білі зображення формату JPEG (розширення файлу зображення повинно бути .jpg). Тренувальний набір – це певна кількість таких зображень будь-якого змісту (проте для більш якісного результату при навчання моделі бажано, щоб один набір мав зображення, в яких є щось спільне), які зібрані в один файл zip-архіву. Кількість файлів зображень в архіві обмежена лише загальним об’ємом цього zip-архіву – 5 Гб. Також файл архіву не повинен містити вкладених папок або додаткових файлів, окрім примірників зображень. Мінімальна кількість зображень на один тренувальний набір – 4 примірника. Проте треба враховувати той факт, що чим більше буде примірників в тренувальному наборі, тим якіснішим буде процес навчання моделі. Тому рекомендована кількість зображень в одному наборі – від 400 примірників. З іншого боку, треба також враховувати, що чим більше зображень в наборі, тим довшим буде процес обробки заданої кількості епох.

Вихідними даними всієї системи є навчені стани моделі та колажі згенерованих зображень. Навчений стан – це файл формату BSON (розширення файлу .bson) – бінарна версія формату JSON. Цей файл містить два об’єкти: генератор та дискримінатор. Кожен з цих об’єктів містить весь набір, адаптованих в результаті циклів навчання, параметрів моделі, а також структури цих нейронних мереж. На даному етапі розробки системи така структура файлу стану моделі сумісна лише з фреймворком Flux в мові програмування Julia. Колаж згенерованих зображень – це файл зображення формату JPEG (розширення файлу .jpg), який побудований із 9 згенерованих зображень поєднаних у квадратний колаж-сітку. Такий файл можна отримати в результаті тестування або тренування моделі.

Важливо також зазначити про нормалізацію примірників зображень для їх використання в процесі навчання моделі. Адже для дискримінатора розмір

вхідних даних повинен бути фіксований, а саме 100 на 100 пікселів. Тому кожен примірник зображення проходить такі етапи перетворень, щоб досягти норми:

1. визначається коротша сторона зображення;
2. обрізаються рівні частини по бокам довшої сторони так, щоб зображення стало квадратним;
3. отримане зображення стискається або навпаки розширюється до розміру сторін 100 на 100 пікселів.

Важливо зауважити, що такий підхід до нормалізації зображень призводить до відкидання деякої частини зображення примірника, тому для вищої якості навчання моделі варто підбирати такі зображення, в яких ключові елементи знаходяться посередині, щоб нейронна мережа могла їх розпізнати. Цей алгоритм реалізований в рамках модуля ядра моделі, а інші модулі не виконують ніяких маніпуляцій зі змістом файлів зображень. Далі наведено код функцій, які забезпечують таку нормалізацію:

```
function crop_img(image)
    width = size(image)[1];
    height = size(image)[2];
    min_size = min(width, height);
    pad = ceil(Int, abs(width - height) / 2);
    from_x = width == min_size ? 1 : pad;
    to_x = width == min_size ? min_size : from_x + min_size - 1;
    from_y = height == min_size ? 1 : pad;
    to_y = height == min_size ? min_size : from_y + min_size - 1;
    return image[from_x:to_x, from_y:to_y]
end
function resize_img(image, scaleFactor)
    img_size = size(image);
    if (typeof(image[1, 1, 1]) === Gray{N0f8})
       image = RGB.(image);
    end
```

```
    img_arr = channelview(image);
    itpR = interpolate(Float32.(img_arr[1,:,:]), BSpline(Constant()));
    itpG = interpolate(Float32.(img_arr[2,:,:]), BSpline(Constant()));
    itpB = interpolate(Float32.(img_arr[3,:,:]), BSpline(Constant()));
    imgR_resized = itpR[1:scaleFactor:img_size[1], 1:scaleFactor:img_size[2]];
    imgG_resized = itpG[1:scaleFactor:img_size[1],
       1:scaleFactor:img_size[2]];
    imgB_resized = itpB[1:scaleFactor:img_size[1], 1:scaleFactor:img_size[2]];
    trimF(a) = a < 0. ? 0. : a > 1. ? 1. : a;
    imgR_resized = trimF.(imgR_resized);
    imgG_resized = trimF.(imgG_resized);
    imgB_resized = trimF.(imgB_resized);
    return [N0f8.(imgR_resized);;; N0f8.(imgG_resized);;;
       N0f8.(imgB_resized)]
end
```

Крім того, вихідні згенеровані зображення також мають такий нормалізований розмір, адже генератор працює в парі з дискримінатором під час навчання моделі, а тому повинен дотримуватись цієї норми. Приклади згенерованих зображень подані в додатку Б.

Ще один важливий аспект, який варто зазначити, – це процес вилучення файлів з zip-архіву. Як відомо, існує два основних типи zip-архіві: звичайний ZIP (розмір такого архіву не може перевищувати 4 Гб) та ZIP64 (розмір такого файлу не може перевищувати 16 ексабайтів, тобто можна сказати, що об’єм даних для цього формату архіву необмежений). Проте в стандартних пакетах мови Julia немає реалізації розпакування архівів формату ZIP64 (вона поки знаходиться на стадії розробки спільнотою програмістів). Тому було розроблено власний алгоритм роботи з таких форматом із застосуванням існуючої бази для роботи з форматом ZIP. Код функцій, які забезпечують розпакування архіву в чотиривимірну матрицю пікселів, яку використовує нейронна мережа в процесі навчання представлено далі:

```
function get_filesize(archive, id)
    size = archive.files[id].uncompressedsize;
    size == 0xFFFFFFFF || return size
    s = archive._io;
    namelen = length(archive.files[id].name);
    seek(s, archive.files[id]._offset);
    skip(s, 4+2+2+2+2+2+4+4+4+2+2+namelen+2+2);
    return Int(read(s, UInt64))
end
function read_jpg_from_zip(file::IO, img_num::Integer, n_features::Integer)
    archive = ZipFile.Reader(file);
    imgs = Array{RGB{N0f8}}(undef, n_features, n_features, 3, 0);
    for i in 1:img_num
       contains(archive.files[i].name, ".jpg") || continue
       @time begin
              bytes = read(archive.files[i], get_filesize(archive, i));
              img = ImageMagick.load_(bytes);
              cropped_img = crop_img(img);
              norm_img = resize_img(cropped_img, size(cropped_img)[1]
                     / n_features);
              imgs = cat(imgs, norm_img; dims=4);
       end
    end
    return imgs
end
```

2.6. Опис розробленої системи

2.6.1. Використані технічні засоби

Для роботи із моделями машинного навчання в AWS є сервіс SageMaker. SageMaker дозволяє розробникам працювати на кількох рівнях абстракції під час

навчання та розгортання моделей машинного навчання. Незалежно від того, який рівень абстракції використовується, розробник може підключити свої моделі ML з підтримкою SageMaker до інших служб AWS, таких як база даних Amazon DynamoDB або сервіс зберігання файлів AWS S3 для зберігання структурованих даних, наборів тренувальних даних та станів моделі після навчання. Для взаємодії з SageMaker розробникам доступний ряд інтерфейсів. По-перше, є веб-API, який віддалено керує екземпляром сервера SageMaker. Хоча веб-API не залежить від мови програмування, яку використовує розробник, Amazon надає прив'язки API SageMaker для кількох мов, включаючи Julia, проте є необхідність в специфічних налаштуваннях для роботи цієї мови на машині SageMaker. Крім того, SageMaker надає керовані екземпляри Jupyter Notebook для інтерактивного програмування SageMaker. Саме Jupyter Notebook разом з мовою програмування Julia буде використовуватись для побудови та навчання моделі. Проте для запуску серверу Julia на базі машини SageMaker необхідно описати скрипт, який буде завантажувати та запускати всі необхідні програми для роботи серверу Julia на базі Jupyter Notebook. Далі представлено shell-скрипт, який завантажує ядро Julia під час створення машини SageMaker:

```
#!/bin/bash
set -e
sudo -u ec2-user -i <<EOF
echo ". /home/ec2-user/anaconda3/etc/profile.d/conda.sh" >> ~/.bashrc
conda create --yes --prefix ~/SageMaker/envs/julia
wget -q https://julialang-s3.julialang.org/bin/linux/x64/1.7/julia-1.7.2-linux-x86
    _64.tar.gz -O - | tar xzf -
cp -R julia-1.7.2/* ~/SageMaker/envs/julia/
mkdir -p ~/SageMaker/envs/julia/etc/conda/activate.d
echo 'export JULIA_DEPOT_PATH=~/SageMaker/envs/julia/depot' >> ~/Sage
    Maker/envs/julia/etc/conda/activate.d/env.sh
echo -e 'empty!(DEPOT_PATH)\npush!(DEPOT_PATH,raw"/home/ec2-user/
    SageMaker/envs/julia/depot")' >> ~/SageMaker/envs/julia/etc/julia/
```

```
startup.jl
conda activate /home/ec2-user/SageMaker/envs/julia
julia --eval 'using Pkg; Pkg.add("IJulia"); using IJulia'
EOF
```

В результаті деяких експериментів було обрано машину SageMaker на базу графічного процесора. В таблиці 2.3 представлено характеристики цієї машини.

Таблиця 2.3

Характеристики машини SageMaker

Тип машини	CPU (ядра)	GPU	ОЗП (Гб)	SSD (Гб)	Ціна за час (USD)
ml.g4dn.xlarge	4	NVIDIA T4	16	5	0,7364

Цей тип обчислювальної машини показав високі значення швидкості обробки при навчання моделі DCGAN – в середньому від 0,5 до 1 секунди на одну епоху.

Для розгортання модулів керування станом моделі та керування наборами тренувальних даних в ідеалі може бути застосовано AWS EC2 або AWS ECS, проте для економії коштів при розробці ці модулі були розгорнуті на базі локального ПК Apple Macbook Pro на процесорі Apple Silicon M1 Pro з ОЗП 16 Гб та SSD 512 Гб.

2.6.2. Використані програмні засоби

Розроблена інформаційна система написана на двох мовах програмування: Julia для модуля ядра моделі DCGAN, Java для модулів керування станами моделі та наборами тренувальних даних. Розробка ядра моделі на Julia велася у веб-середовищі розробки Jupyter Notebook. Розробка обслуговуючих модулів велася в інтегрованому середовищі розробки IntelliJ IDEA Community Edition. База даних системи була побудована на базі AWS DynamoDB та AWS S3.

2.6.3. Виклик та завантаження програми

Система запускається через запуск двох модулів на Java. Попередньо повинні бути створені два кошика AWS S3 (їх назви «gan-datasets» та «gan-state»), завантажена БД DynamoDB та створена одна машина SageMaker типу ml.g4dn.xlarge з назвою «gan» та конфігурацією життєвого циклу із зазначеним в підрозділі 2.6.1 скриптом на етап створення машини (в локальну пам’ять машини повинні бути встановлені відповідні скрипти для навчання та тестування моделі).

2.6.4. Опис інтерфейсу користувача

У розроблюваної інформаційної системи немає графічного інтерфейсу користувача, адже ця система є суто back-end застосунком. Проте для зручності використання API, для його тестування, була застосована бібліотека для побудови документації для REST API – Swagger UI.

Swagger – це програма з відкритим вихідним кодом, яка підтримується великою екосистемою інструментів, які допомагають розробникам проектувати, створювати, документувати та використовувати веб-сервіси на базі REST API. Найбільш популярним є Swagger UI, який виявився найбільш зручним для побудови документації для REST API. Набір інструментів Swagger включає також підтримку автоматизованої документації, генерації коду та тестових прикладів на основі, наприклад, шару контролерів в Java Spring застосунку. Swagger розвивається за підтримки SmartBear Software.

Далі представлено зовнішній вигляд документації Swagger, яка згенерована за допомогою спеціальної бібліотеки springfox на основі написаного Java-коду для шару контролерів. Зовнішній вигляд згенерованої інтерактивної документації представлено на рисунках 2.5 та 2.6.

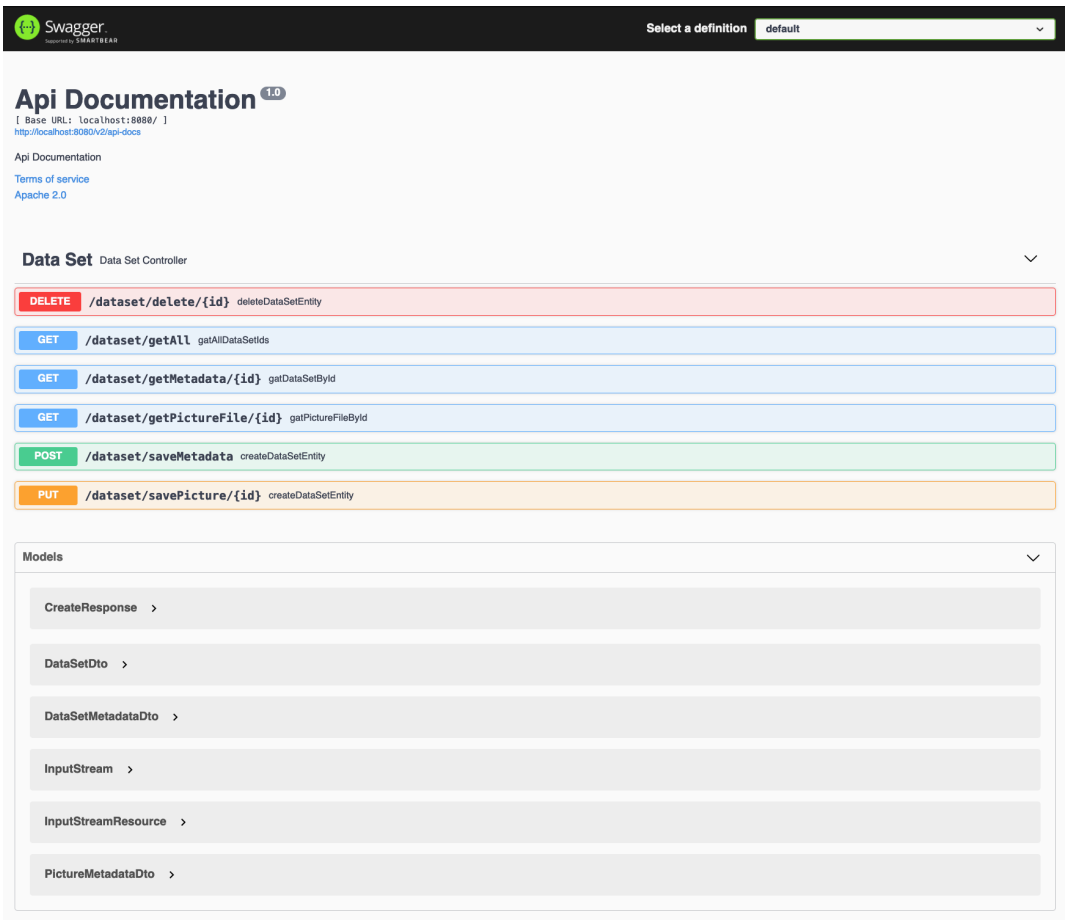

Рис. 2.5. API для модуля DatasetManager

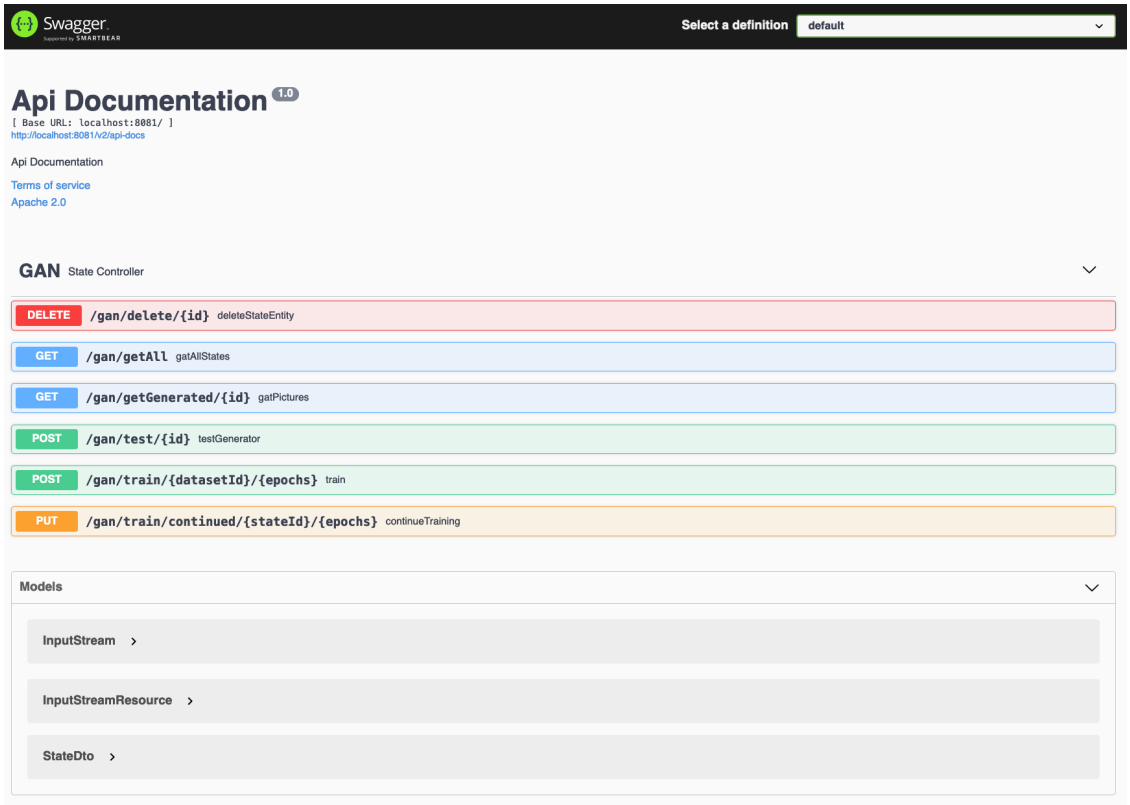

Рис. 2.6. API для модуля StateManager

Далі на рис. 2.7-2.9 представлені методи GET для DataSetManager.

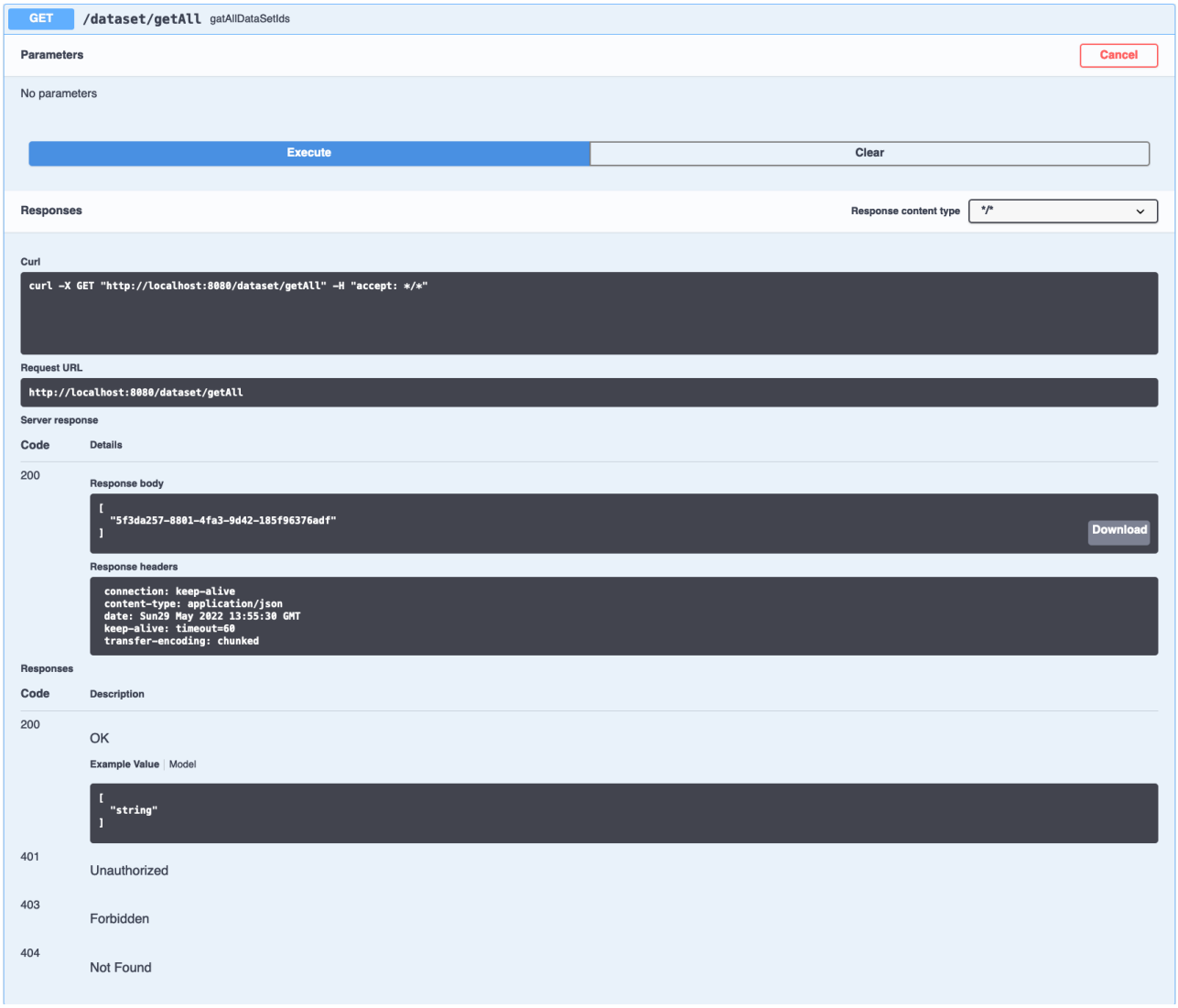

Рис. 2.7. Метод getAll

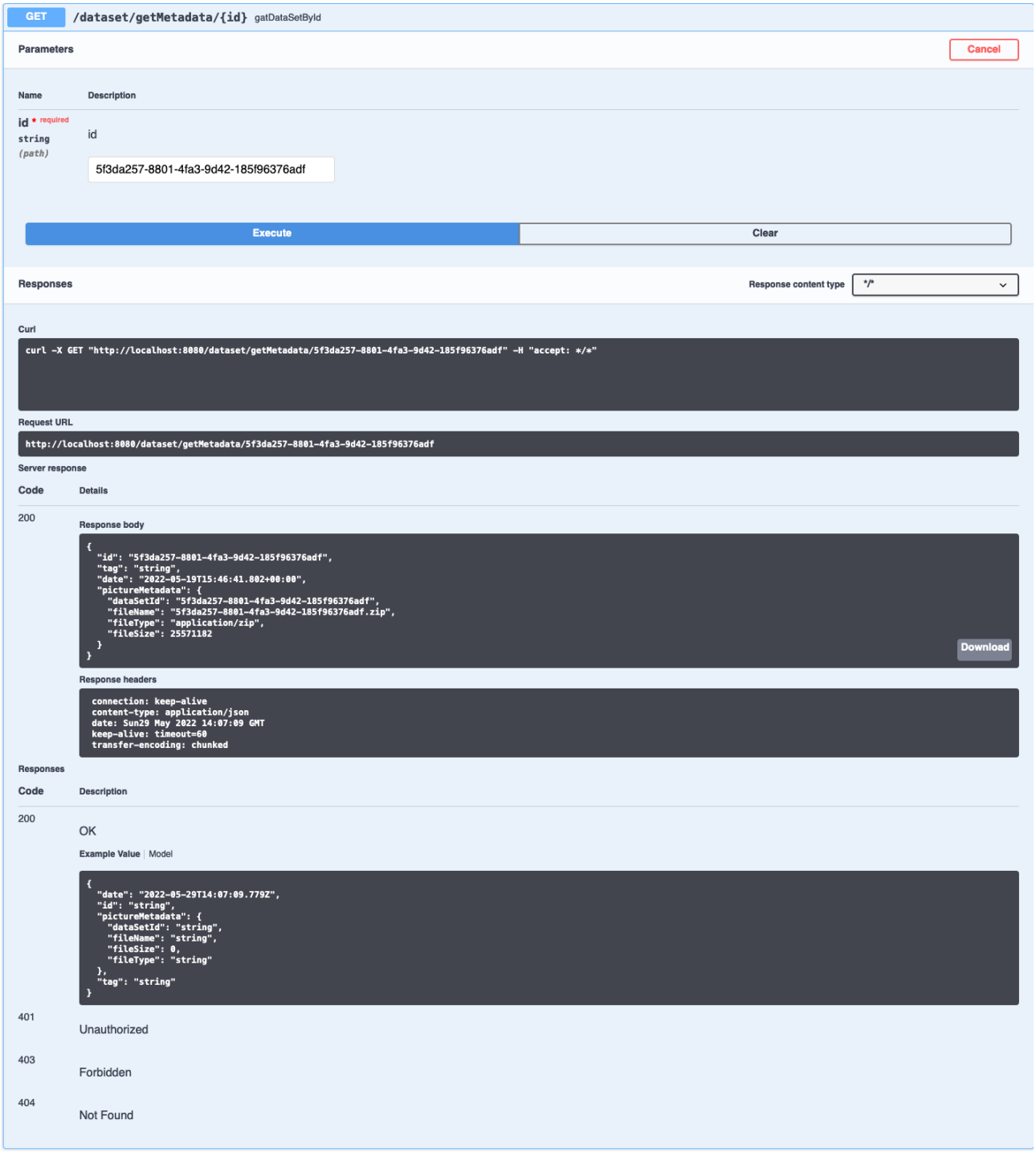

Рис. 2.8. Метод getMetadata

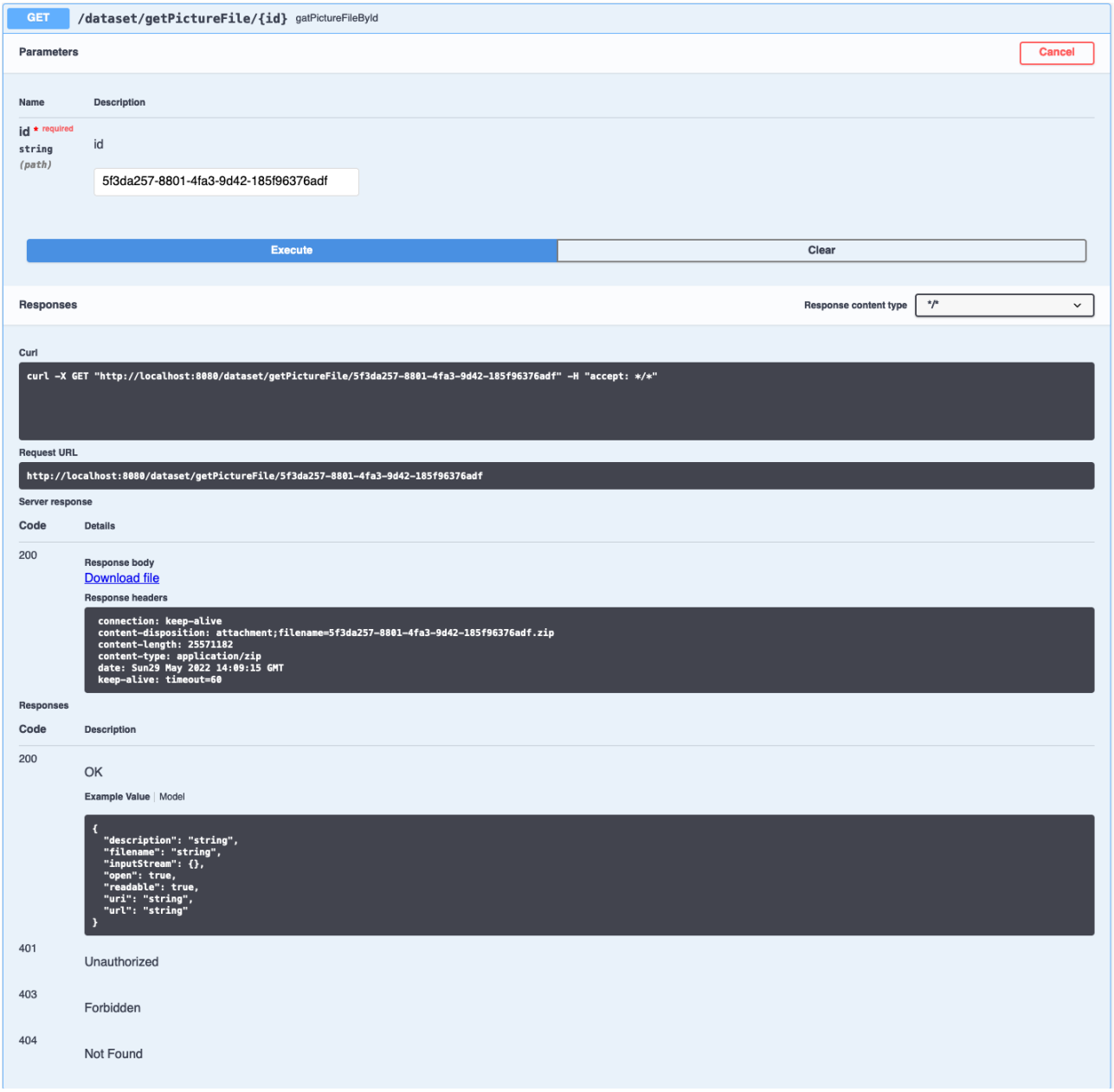

Рис. 2.9. Метод getPictureFile

Далі на рис. 2.10 представлений метод POST для DataSetManager.

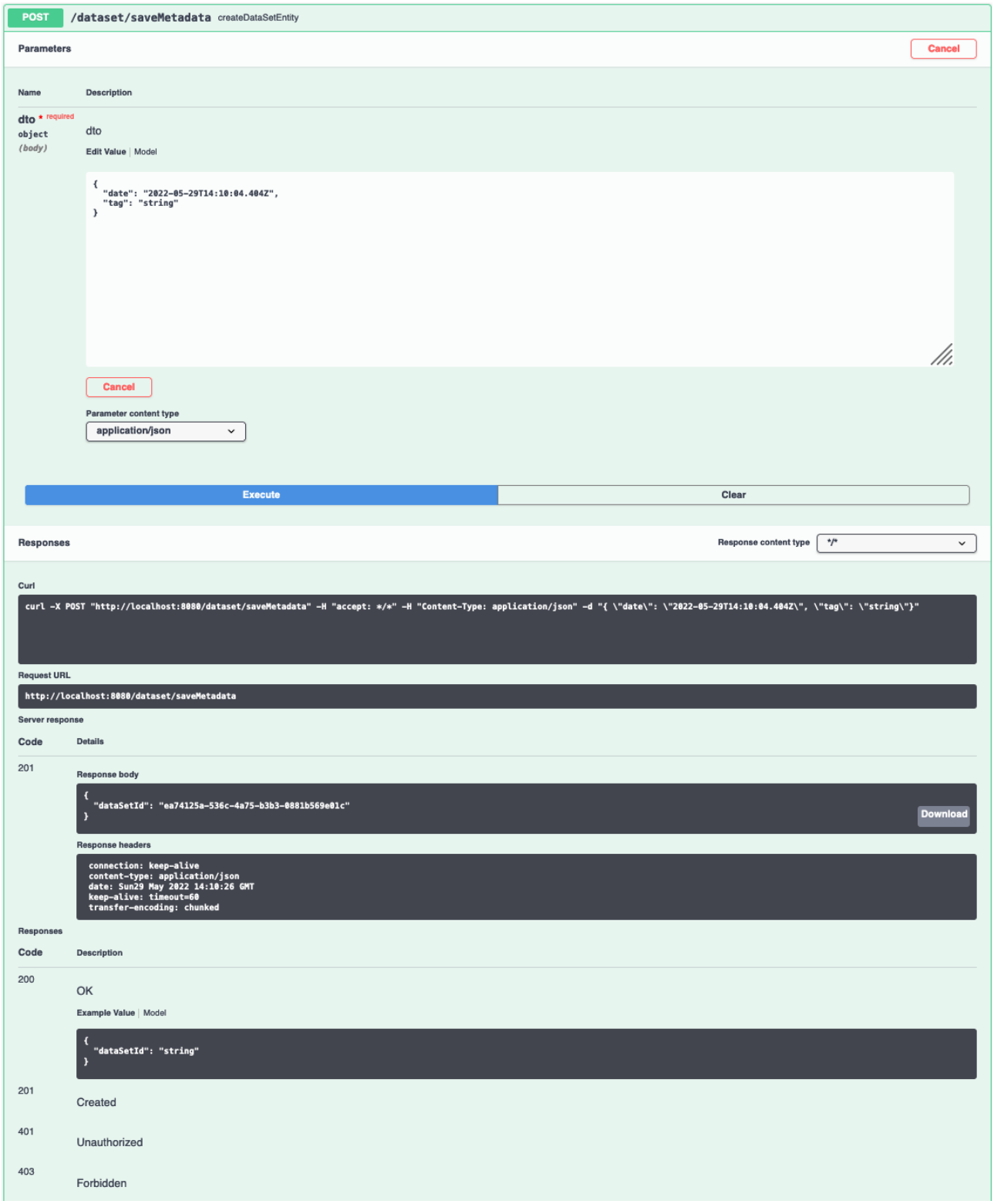

Рис. 2.10. Метод saveMetadata

Далі на рис. 2.11 представлений метод PUT для DataSetManager.

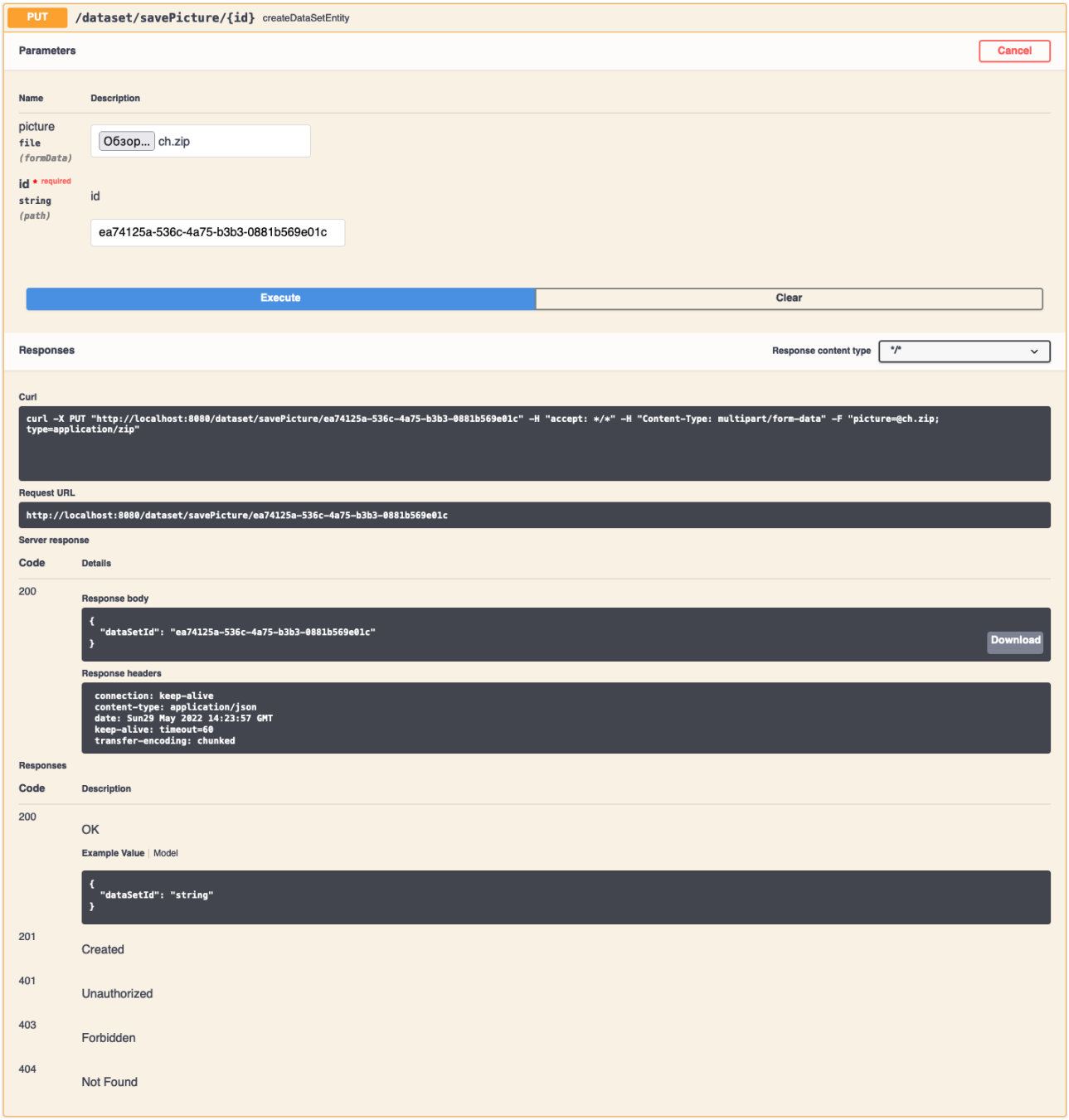

Рис. 2.11. Метод savePicture

Далі на рис. 2.12 представлений метод DELETE для DataSetManager.

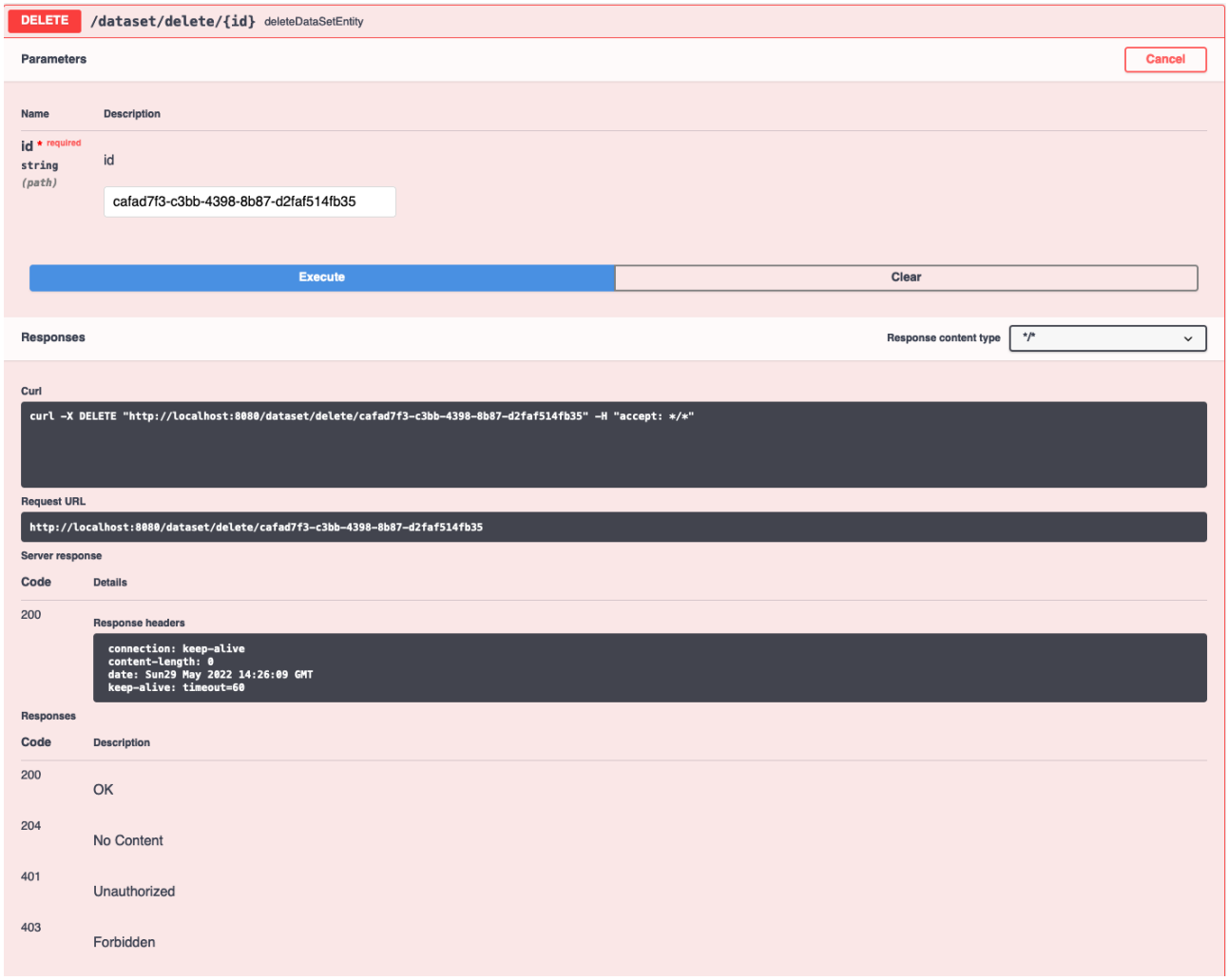

Рис. 2.12. Метод delete

Далі на рис. 2.13 та 2.14 представлений методи GET для StateManager.

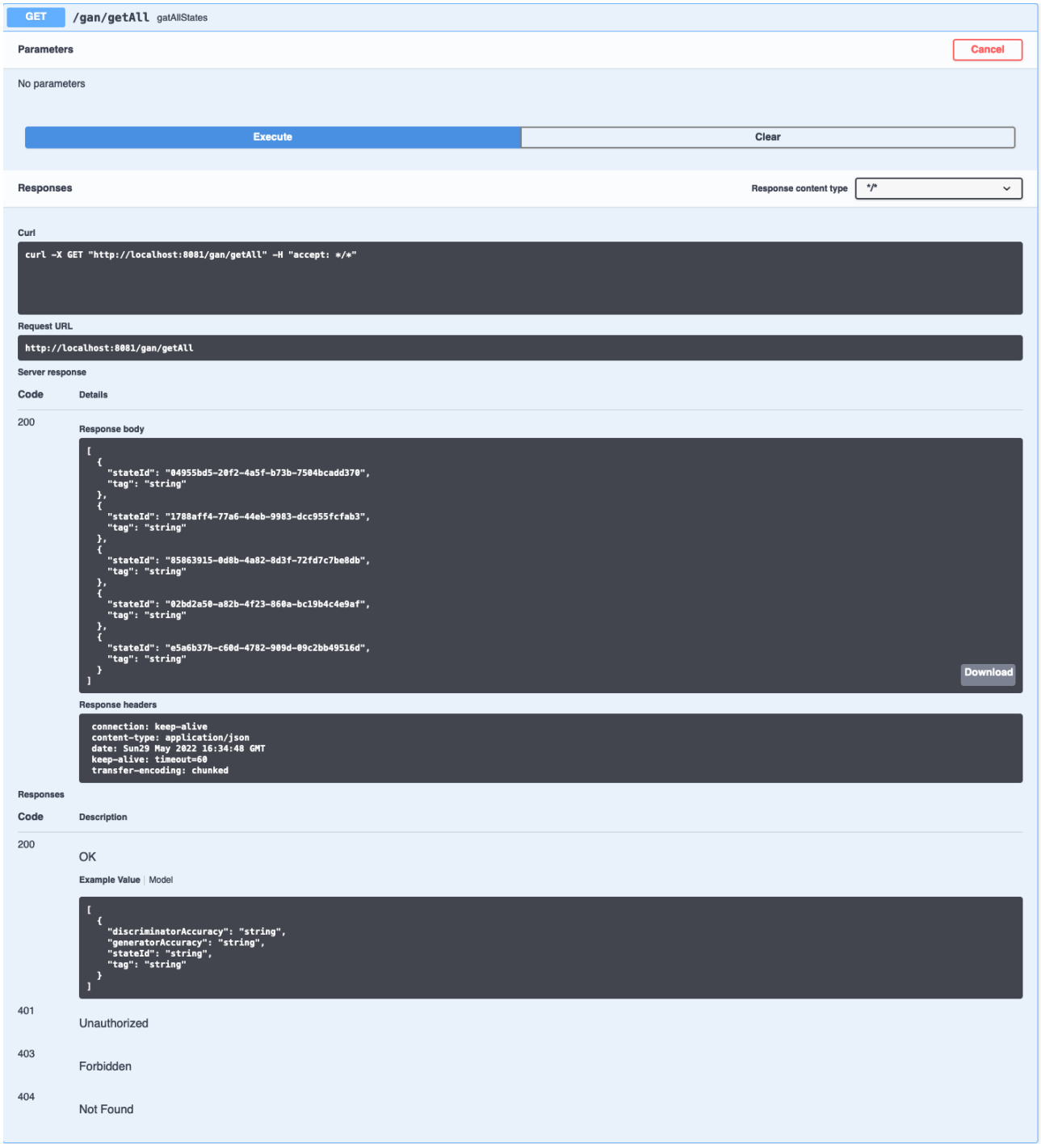

Рис. 2.13. Метод getAll

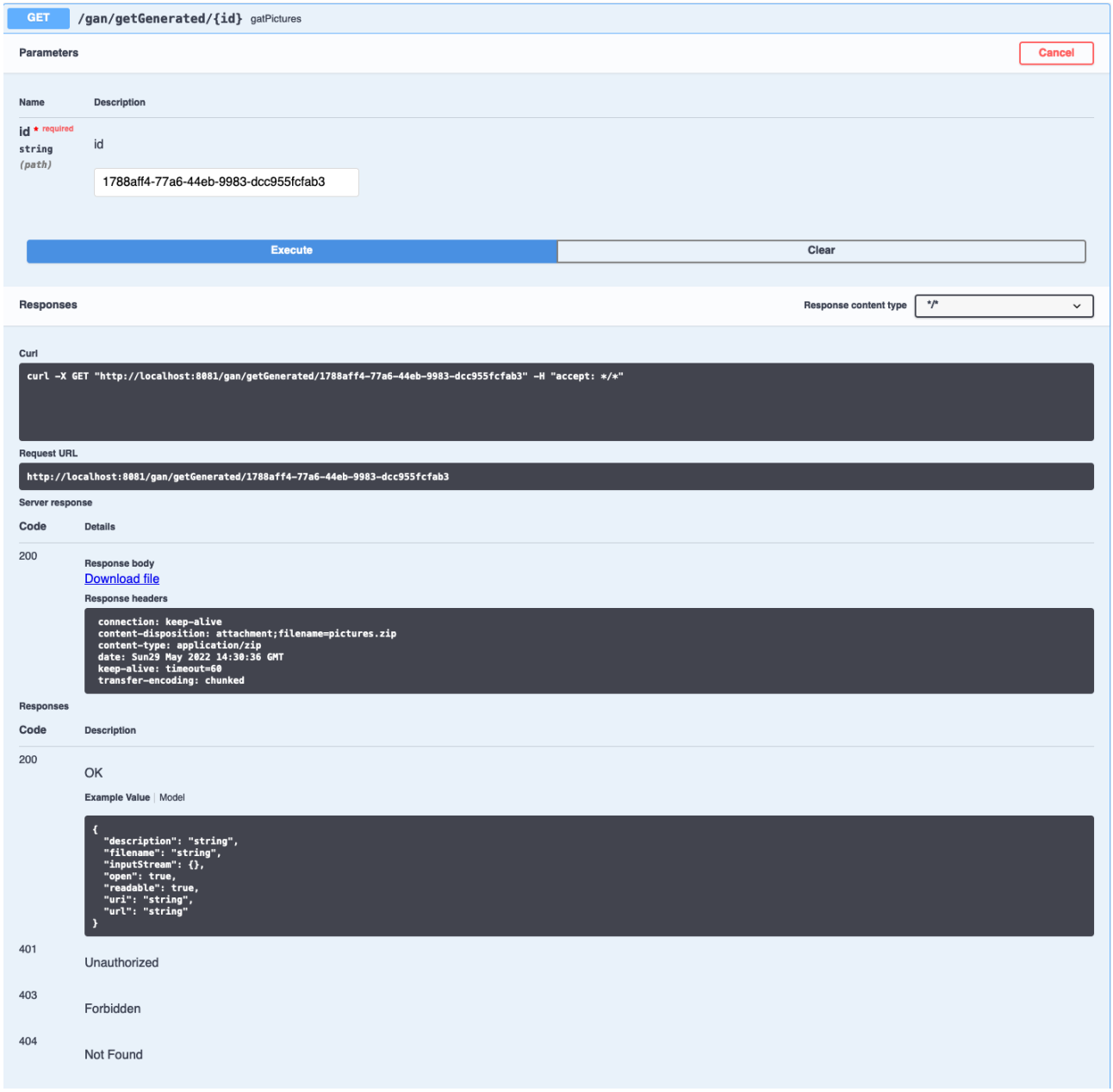

Рис. 2.14. Метод getGenerated

Далі на рис. 2.15 й 2.16 представлений методи POST для StateManager.

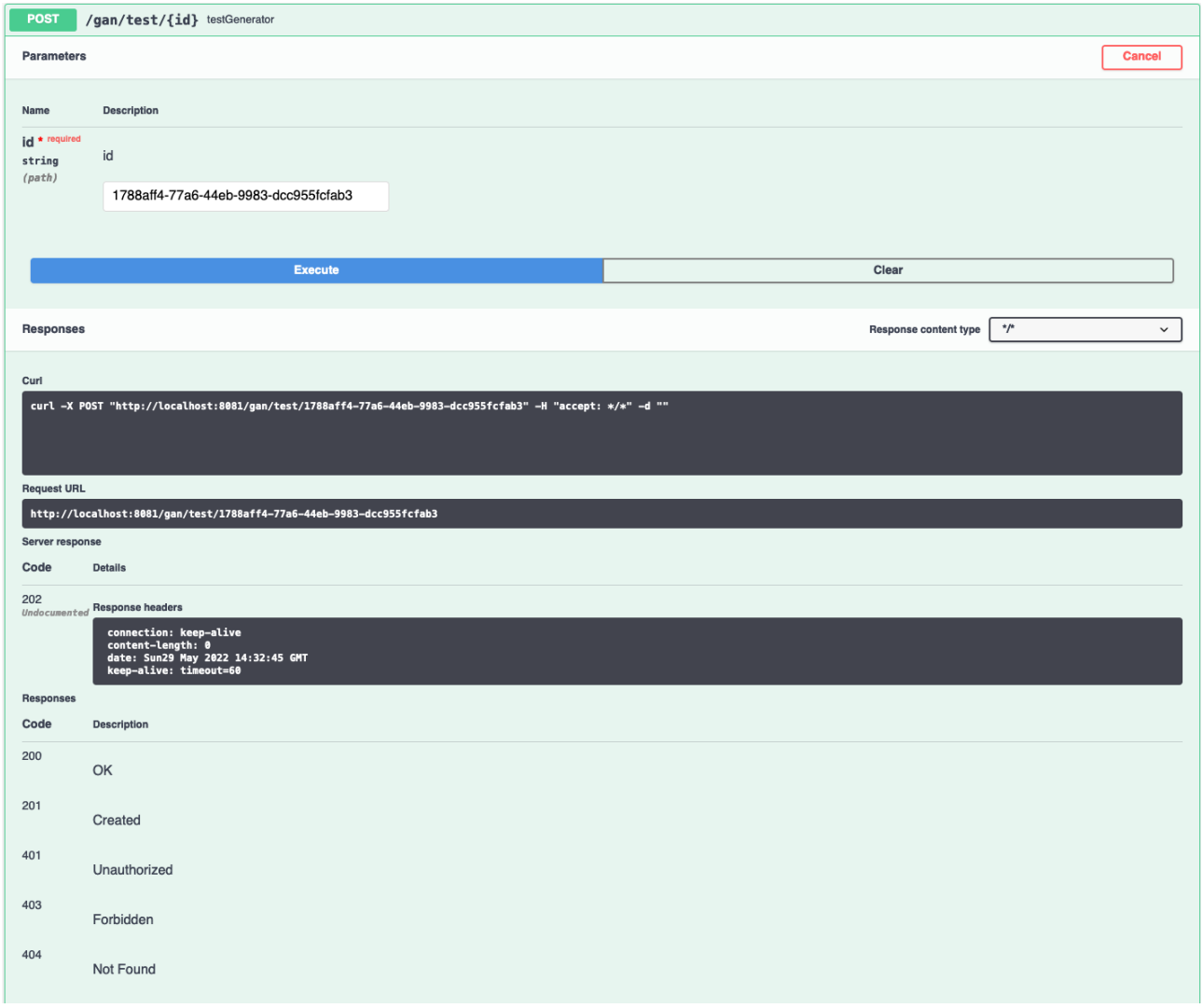

Рис. 2.15. Метод test

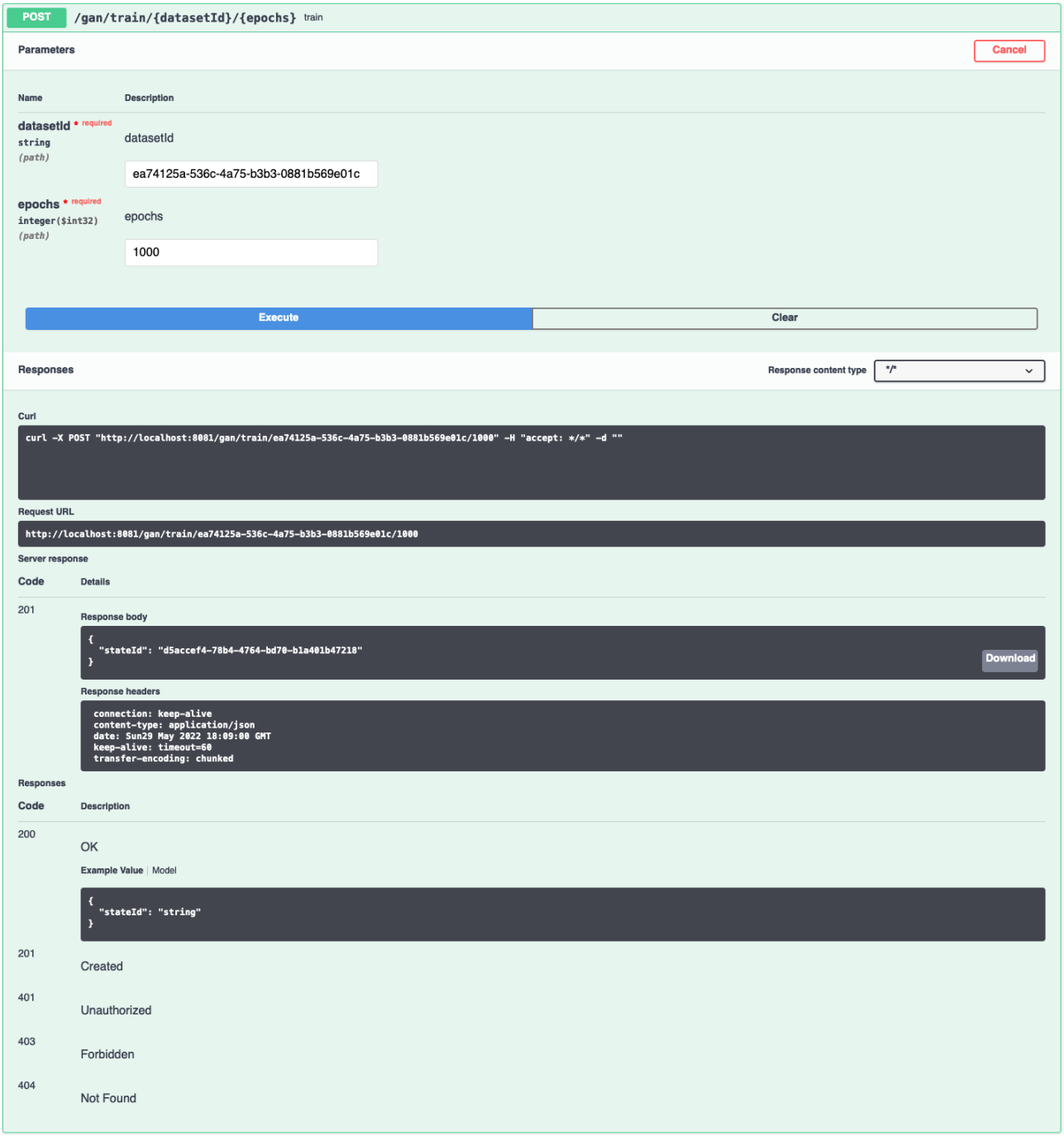

Рис. 2.16. Метод train

Далі на рис. 2.17 представлений методи PUT для StateManager.

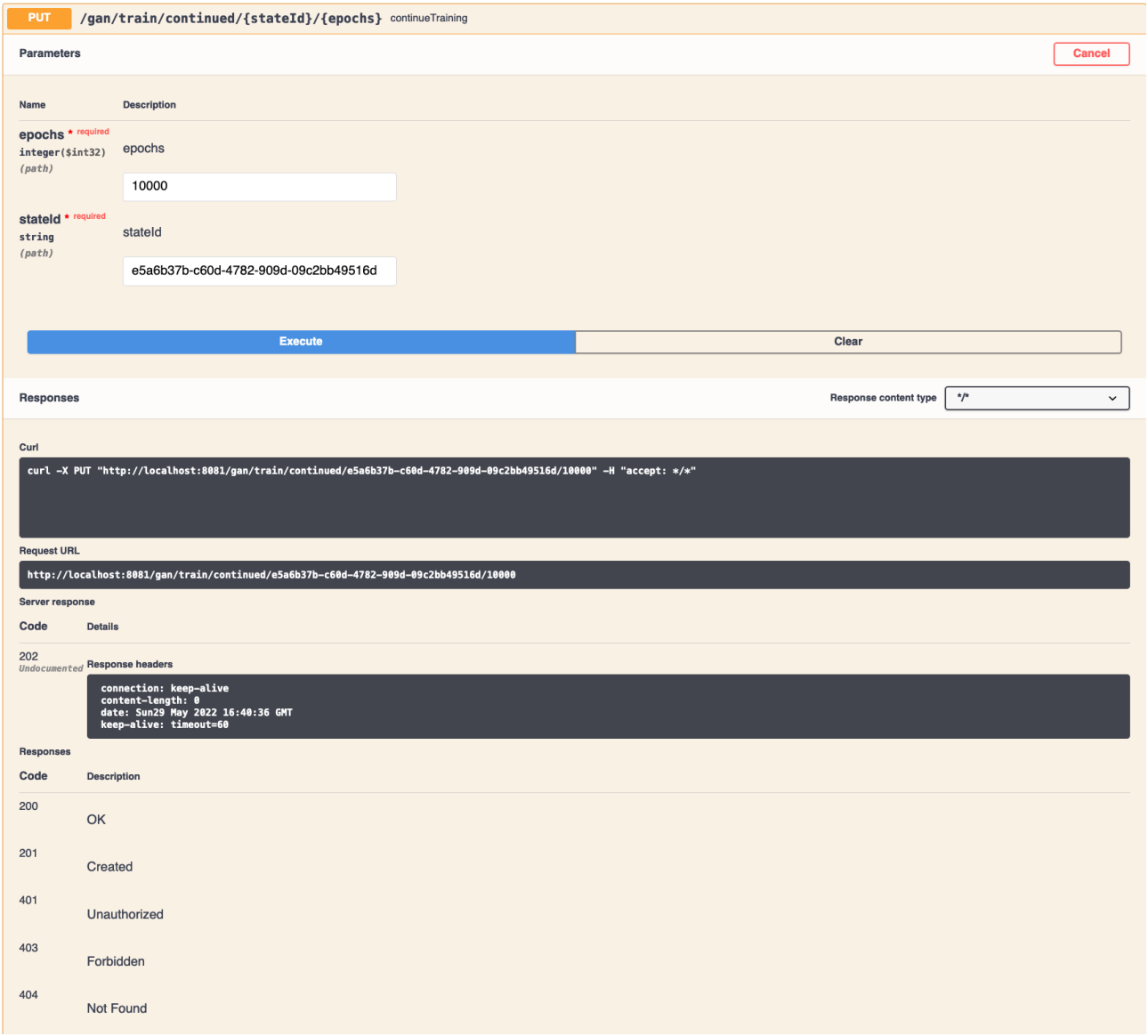

Рис. 2.17. Метод train/continued

Далі на рис. 2.18 представлений методи DELETE для StateManager.

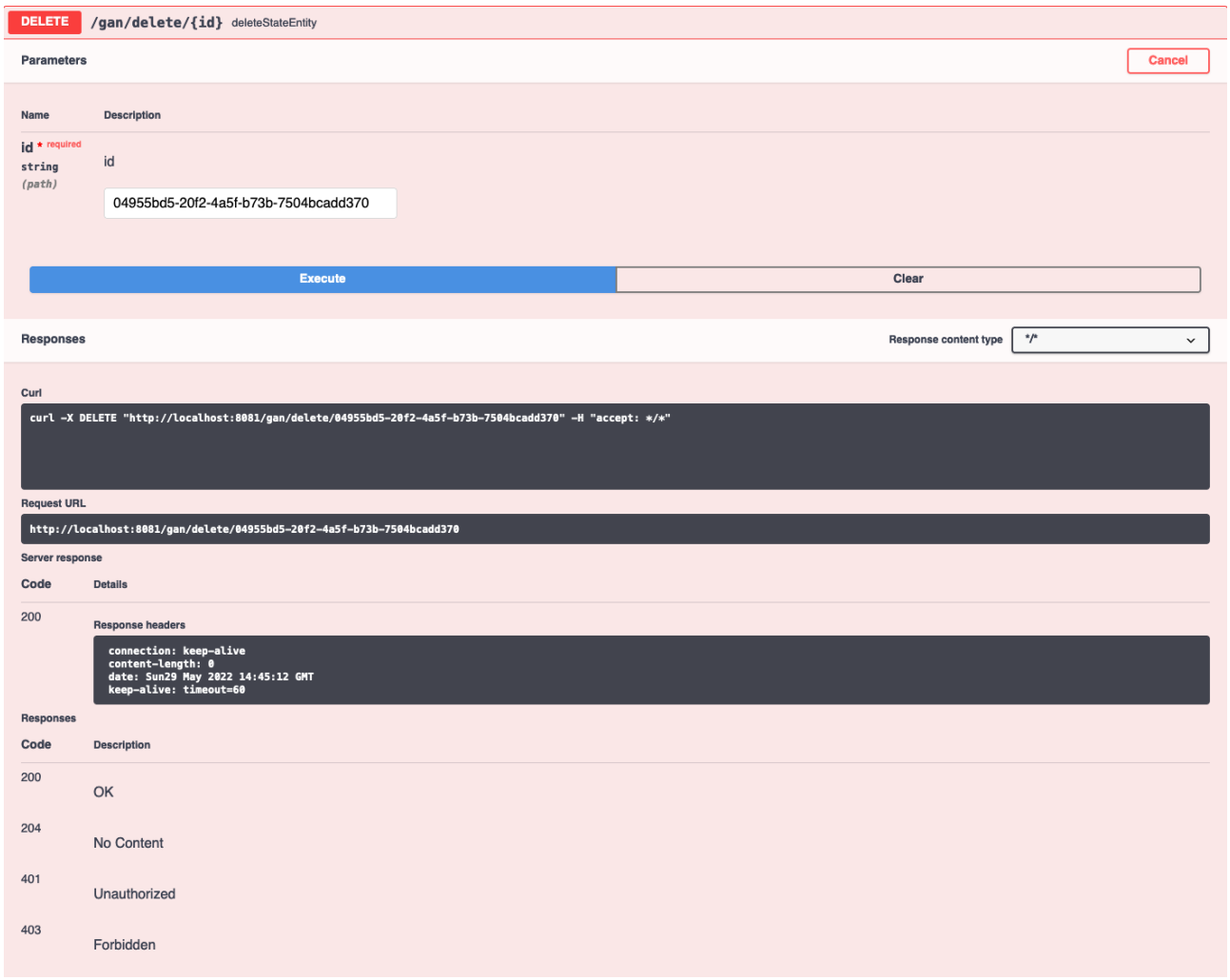

Рис. 2.18. Метод delete

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості розробки програмного продукту

Вхідні дані:

– передбачувана калькість операторів – 1250;

– коефіцієнт складності програми – 1,85;

– коефіцієнт корекції програми в ході її розробки – 0,07;

– коефіцієнт збільшення витрат праці через недостатній опис задачі – 1,2;

– коефіцієнт кваліфікації програміста – 1,2;

– кількість розробників – 1;

– вартість машино-години ЕОМ – 21,7 грн/год (вартість роботи однієї машини SageMaker типу ml.g4dn.xlarge) [21];

– годинна заробітна плата програміста – 350 грн/год [22].

Нормування праці в процесі створення програмного забезпечення істотно ускладнено тому, що робота програміста має творчий характер. Тому трудомісткість розробки програмного продукту може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість можна рознахувати за такою формулою:

$$t = t_o + t_д + t_a + t_п + t_{налаг} + t_{док}, \quad (3.1)$$

де t – загальна трудомісткість розробки;

t_o – витрати праці на підготовку й опис поставленої задачі (приймається значення 50);

$t_д$ – витрати праці на дослідження алгоритму рішення задачі;

t_a – витрати праці на розробку блок-схеми алгоритму;

$t_п$ – витрати праці на програмування алгоритму;

$t_{налаг}$ – витрати праці на налагодження програми на ЕОМ;

$t_{док}$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів (підпрограм) в розроблюваній програмі.

Умовне число операторів (підпрограм) розраховується за формулою:

$$Q = q * C * (1 + p), \tag{3.2}$$

де q – передбачувана кількість операторів (підпрограм);

C – коефіцієнт складності програми;

p – коефіцієнт корекції програми в ході її розробки.

Витрати праці на дослідження алгоритмів вирішення задачі визначається з урахуванням уточнення опису та кваліфікації програміста. Формула розрахунку:

$$t_д = \frac{Q * B}{(75..85) * k}, \tag{3.3}$$

де B – коефіцієнт збільшення витрат праці через недостатній опис задачі;

k – коефіцієнт кваліфікації програміста, обумовлений досвідом роботи по даній спеціалізації.

За формулами (3.2) та (3.3) обчислюєм витрати праці на дослідження та визначення алгоритмів рішення завдання:

$$Q = 1250 * 1{,}85 * (1 + 0{,}07) = 2474 \text{ людино-годин}, \tag{3.4}$$

$$t_д = \frac{2474 * 1{,}2}{85 * 1{,}2} = 29 \text{ людино-годин}. \tag{3.5}$$

Витрати праці на розробку блок-схеми алгоритму розраховуються так:

$$t_a = \frac{Q}{(20..25) * k} \tag{3.6}$$

За формулою (3.6) та значенням параметру Q, який розрахований в (3.4), обчислюєм витрати праці на розробку алгоритму:

$$t_a = \frac{2474}{22 * 1,2} = 94 \text{ людино-годин.} \quad (3.7)$$

Витрати праці на складання програми алгоритму розраховуються так:

$$t_п = \frac{Q}{(20..25) * k} \quad (3.8)$$

За формулою (3.8) та значенням параметру Q, який розрахований в (3.4), обчислюєм витрати праці на програмування:

$$t_п = \frac{2474}{20 * 1,2} = 103 \text{ людино-годин.} \quad (3.9)$$

Витрати праці на налагодження програми на ЕОМ розраховуються двома формулами, а саме:

– за умови автономного налаготдження одного завдання:

$$t_{налаг} = \frac{Q}{(4..5) * k}; \quad (3.10)$$

– за умови комплексного налаготдження всього завдання:

$$t^k_{налаг} = 1,5 * t_{налаг} \quad (3.11)$$

За формулами (3.10) та (3.11) обчислюєм витрати праці на налагодження:

$$t_{налаг} = \frac{2474}{4 * 1,2} = 515 \text{ людино-годин,} \quad (3.12)$$

$$t^k_{налаг} = 1,5 * 515 = 773 \text{ людино-годин.} \quad (3.13)$$

Витрати праці на підготовку документації розраховуються так:

$$t_{док} = t_{др} + t_{до}, \quad (3.14)$$

де $t_{др}$ – трудомісткість підготовки матеріалів;

$t_{до}$ – трудомісткість редагування та оформлення документації.

Трудомісткість підготовки матеріалів для складання документації розраховується за формулою:

$$t_{др} = \frac{Q}{(15..20) * k} \quad (3.15)$$

Трудомісткість редагування, оформлення документації розраховується так:

$$t_{до} = 0{,}75 * t_{др} \quad (3.16)$$

За формулами (3.15), (3.16) та (3.14) обчислюєм витрати праці на документацію:

$$t_{др} = \frac{2474}{20 * 1{,}2} = 103 \text{ людино-годин}, \quad (3.17)$$

$$t_{до} = 0{,}75 * 103 = 77 \text{ людино-годин}, \quad (3.18)$$

$$t_{док} = 103 + 77 = 180 \text{ людино-годин}. \quad (3.19)$$

Тепер розраховуємо загальну трудомісткість розробки програмного продукту на основі формули (3.1) та на основі результатів, отриманих в (3.5), (3.7), (3.9), (3.13) та (3.19):

$$t = 50 + 29 + 94 + 103 + 773 + 180 = 1229 \text{ людино-годин}. \quad (3.20)$$

3.2. Розрахунок вартості розробки програмного продукту

Витрати на створення програмного продукту включають витрати на заробітню плату розробників програми та витрати мишинного часу, який необхідний на налагодження програми на ЕОМ. Формула виглядає так:

$$К_{ПЗ} = З_{ЗП} + З_{Мч}, \quad (3.21)$$

де $К_{ПЗ}$ – вартість створення програми;

$З_{ЗП}$ – заробітня плата розробникам;

$З_{Мч}$ – витрати машинного часу.

Заробітна плата розробників визначається за формулою:

$$З_{ЗП} = t * С_{пр}, \quad (3.22)$$

де t – загальна трудомісткість, яка була розрахована в попередьому підрозділі;

$С_{пр}$ – середня годинна заробітня плата програміста.

За формулою (3.22) та на основі розрахованої в попередньому підрозділі трудомісткості (3.20) обчислюєм заробітню плату програміста:

$$З_{ЗП} = 1229 * 350 = 430150 \text{ грн.} \quad (3.23)$$

Вартість машинного часу, необхідного для налагодження програми розраховується за такою формулою:

$$З_{Мч} = t_{налаг} * С_{М}, \quad (3.24)$$

де $t_{налаг}$ – витрати праці на налагодження програми на ЕОМ;

$С_{м}$ – вартість машино-години ЕОМ.

За формулою (3.24) та на основі витрати праці на налагодження, які були розраховані в (3.13), обчислюєм вартість машинного часу:

$$З_{МЧ} = 773 * 21{,}7 = 16774 \text{ грн.} \quad (3.25)$$

Тепер розраховуємо за формулою (3.21) вартість розробки програмного продукту, що є одноразовими капітальними витратами:

$$К_{ПЗ} = 430150 + 16774 = 446924 \text{ грн.} \quad (3.26)$$

Очікуваний період створення програми розраховується так:

$$Т = \frac{t}{B_k * F_p}, \quad (3.27)$$

де B_k – кількість розробників;

F_p – місячний фонд робочого часу (приймаємо 176 годин).

За формулою (3.24) обчислюєм очікуваний період розробки програми:

$$Т = \frac{1229}{1 * 176} = 7 \text{ міс.} \quad (3.28)$$

Висновок: було визначено трудомісткість розробки описаної інформаційної системи (1229 людино-годин), розраховано вартість роботи програміста над цим проектом (446924 грн.) та підраховано період розробки (7 місяців).

ВИСНОВКИ

В результаті роботи над кваліфікаційною роботою була розроблена інформаційна система, яка дозволяє тренувати та тестувати модель машинного навчання генеративно-змагальної нейронної мережі. Ця модель реалізовує концепцію машинного мистецтва. Розроблена система дозволяє виконувати операції, які забезпечують досягнення зазначених цілей, зокрема:

- завантажувати та зберігати набори тренувальних даних;
- запускати процес тренування моделі на збереженому наборі;
- зберігати різні навчені стани моделі;
- тестувати роботу певного стану моделі;
- отримувати згенеровані моделлю зображення.

Система розгортається на базі сервісу хмарних обчислень AWS, що дозволило значно оптимізувати та прискорити процес навчання моделі, адже хмарні сервіси дозволяють виконувати код на високошвидкісних апаратних засобах, таких як машини SageMaker з графічним процесором, який спеціально призначений для роботи з моделями машинного навчання.

Дану систему можуть використовувати науковці, які вивчають феномен творчості та застосовують відповідні математичні методи та моделі для імітації цього процесу. Також система буде корисна для аналітиків даних, які знаходять певні закономірності в графічній інформації. Крім того, розробники ігрових застосунків можуть використовувати цю інформаційну систему для генерування унікальних ігрових об’єктів. Загалом, розроблена система є back-end застосунком, тому з нею можуть також спілкуватися інші системи через розроблений інтерфейс, використовуючи створену інтерактивну документацію.

Як і будь-який програмний продукт, створена система не є досконалою і максимально повною. Тому в залежності від відгуків користувачів можуть бути розроблені додаткові шляхи оптимізації роботи системи, також можуть бути розширені можливості системи та впроваджені певні нові функції. Наприклад, є перспектива розширення можливостей роботи з наборами даних, зокрема

групування даних по якомусь загальному показнику (певні заготовки для розвитку в цьому напрямку вже закладені в коді). Також в перспективі можна розширити перелік налаштувань для моделі, які може вказувати користувач. На додачу, можна побудувати повноцінний інтерфейс користувача, якщо в цьому буде необхідність. Отже, перспективи подальшого розвитку та модернізації системи мають декілька конкретних напрямків.

В економічному розділі були розраховані трудомісткість розробки системи (1229 людино-годин), вартість роботи по створенню програмного продукту (446924 грн.), період розробки системи до поточного стану (7 місяців).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Tom M. Mitchell. Machine Learning. – McGraw-Hill Science/Engineering/ Math, 1997. – 432 с.

2. R.A. Fisher. Опис бази даних Iris. Стаття на архівному сайті Machine Learning Repository. [Електронний ресурс]. – Режим доступу: https://archive.ics.uci.edu/ml/datasets/iris.

3. Anna Jordanous. Основні поняття та призначення машинної творчості. Стаття на сайті The creativity post. [Електронний ресурс]. – Режим доступу: https://www.creativitypost.com/science/what_is_computational_creativity.

4. Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. Generative Adversarial Nets. – Архів наукових статей arXiv, 2014. – 9 с. arXiv: 1406.2661.

5. Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, Yoshua Bengio. Maxout Networks. – Архів наукових статей arXiv, 2013. – 9 с. arXiv: 1302.4389v4.

6. Alec Radford, Luke Metz, Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. – Архів наукових статей arXiv, 2016. – 16 с. arXiv: 1511.06434.

7. Свідчення спеціалістів, які підтвердили ефективність застосування функції активації LeakyReLU в моделі GAN. [Електронний ресурс]. – Режим доступу: https://www.reddit.com/r/MLQuestions/comments/c96mci/why_do_gans_ only_work_with_leaky_relu.

8. Ian Goodfellow, Yoshua Bengio, Aaron Courville. Deep Learning. – The MIT Press, 2016. – 800 с.

9. Sergey Ioffe, Christian Szegedy. Batch Normalization: accelerating deep network training by reducing internal covariate shift. – Архів наукових статей arXiv, 2015. – 11 с. arXiv: 1502.03167.

10. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. – Journal of Machine Learning Research vol. 15, 1929−1958, 2014. – 30 с.

11. Daniel Godoy. Опис розрахунку функції помилки binary cross-entropy. [Електронний ресурс]. – Режим доступу: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a.

12. David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams. Nature vol. 323, 533–536, 1986. – 3 с. DOI: 10.1038/323533a0.

13. Віталій Бушаєв. Опис оптимізатора SGD з моментом. Стаття на сайті Towards data science. [Електронний ресурс]. – Режим доступу: https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d.

14. Віталій Бушаєв. Опис оптимізатора RMSprop. Стаття на сайті Towards data science. [Електронний ресурс]. – Режим доступу: https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a.

15. Diederik P. Kingma, Jimmy Lei Ba. ADAM: a method for stochastic optimization. – Архів наукових статей arXiv, 2017. – 15 с. arXiv: 1412.6980.

16. Віталій Бушаєв. Опис оптимізатора ADAM. Стаття на сайті Towards data science. [Електронний ресурс]. – Режим доступу: https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c.

17. Jeff Bezanson, Stefan Karpinski, Viral B. Shah, Alan Edelman. Основні причини створення мови програмування Julia. Стаття в розділі блогів на офіційному сайті проекту Julia. [Електронний ресурс]. – Режим доступу: https://julialang.org/blog/2012/02/why-we-created-julia.

18. Jeff Bezanson, Stefan Karpinski, Viral B. Shah, Alan Edelman. Документація мови програмування Julia. The Julia Project, 2022. – 1378 с. [Електронний ресурс]. – Режим доступу: https://raw.githubusercontent.com/JuliaLang/docs.julialang.org/assets/julia-1.7.3.pdf.

19. Документація для фреймворку Flux. Офіційний сайт проекту. [Електронний ресурс]. – Режим доступу: https://fluxml.ai/Flux.jl/stable.

20. Fatima Hasan. Опис шаблону проектування IoC. Стаття на сайті Educative. [Електронний ресурс]. – Режим доступу: https://www.educative.io/edpresso/what-is-inversion-of-control.

21. Ціни обчислювальних машин різного типу для сервісу SageMaker. Офіційний сайт Amazon Web Services. [Електронний ресурс]. – Режим доступу: https://aws.amazon.com/sagemaker/pricing.

22. Середня заробітна плата розробника на мові програмування Java в Україні. Сайт вакансій Work.ua. [Електронний ресурс]. – Режим доступу: https://www.work.ua/salary-програміст+java.

ДОДАТОК А

КОД ПРОГРАМИ

А.1. Код модуля ядра моделі

Код для тестування моделі на Julia:

```
using CUDA
using Flux
using Random
using FileIO
using Images
using ImageMagick
using BSON
using AWSS3
using AWS
aws = global_aws_config(; region="us-east-1")
if s3_exists(aws, "gan-state", "dcgan.bson")
    file = s3_get(aws, "gan-state", "dcgan.bson");
    generator = gpu(BSON.load(IOBuffer(file))[:generator])
    n = randn(Float32, 100, 9) |> gpu;
    a = 0.5f0 * cpu(generator(n)) .+ 0.5f0;
    to_save_1 = cat(a[:, :, :, 1], a[:, :, :, 2], a[:, :, :, 3]; dims=2);
    to_save_2 = cat(a[:, :, :, 4], a[:, :, :, 5], a[:, :, :, 6]; dims=2);
    to_save_3 = cat(a[:, :, :, 7], a[:, :, :, 8], a[:, :, :, 9]; dims=2);
    to_save = cat(to_save_1, to_save_2, to_save_3; dims=1);
    img = colorview(RGB, permutedims([N0f8.(to_save[:, :, 1]);;; N0f8.(to_save[:, :, 2]);;; N0f8.(to_save[:, :, 3])],
(3,1,2)))
    s3_put(aws, "gan-state", "dcgan.jpg", getblob(image2wand(img), "JPEG"))
end
```

Код для тренування моделі на Julia:

```
using CUDA
using Flux.Data: DataLoader
using Flux
using Zygote
using Random
using FileIO
using Images
using ImageMagick
using ZipFile
using Interpolations
using BSON
using JSON
using AWSS3
using AWS
aws = global_aws_config(; region="us-east-1")
function crop_img(image)
    width = size(image)[1];
    height = size(image)[2];
    min_size = min(width, height);
    pad = ceil(Int, abs(width - height) / 2);
    from_x = width == min_size ? 1 : pad;
    to_x = width == min_size ? min_size : from_x + min_size - 1;
    from_y = height == min_size ? 1 : pad;
    to_y = height == min_size ? min_size : from_y + min_size - 1;
    return image[from_x:to_x, from_y:to_y]
```

```
end
function resize_img(image, scaleFactor)
    img_size = size(image);
    if (typeof(image[1, 1, 1]) === Gray{N0f8})
        image = RGB.(image);
    end
    img_arr = channelview(image);
    itpR = interpolate(Float32.(img_arr[1,:,:]), BSpline(Constant()));
    itpG = interpolate(Float32.(img_arr[2,:,:]), BSpline(Constant()));
    itpB = interpolate(Float32.(img_arr[3,:,:]), BSpline(Constant()));
    imgR_resized = itpR[1:scaleFactor:img_size[1], 1:scaleFactor:img_size[2]];
    imgG_resized = itpG[1:scaleFactor:img_size[1], 1:scaleFactor:img_size[2]];
    imgB_resized = itpB[1:scaleFactor:img_size[1], 1:scaleFactor:img_size[2]];
    trimF(a) = a < 0. ? 0. : a > 1. ? 1. : a;
    imgR_resized = trimF.(imgR_resized);
    imgG_resized = trimF.(imgG_resized);
    imgB_resized = trimF.(imgB_resized);
    return [N0f8.(imgR_resized);;; N0f8.(imgG_resized);;; N0f8.(imgB_resized)]
end
function get_file_num_in_zip(file::IO)
    archive = ZipFile.Reader(file);
    return size(archive.files)[1]
end
function get_filesize(archive, id)
    size = archive.files[id].uncompressedsize;
    size == 0xFFFFFFFF || return size
    s = archive._io;
    namelen = length(archive.files[id].name);
    seek(s, archive.files[id]._offset);
    skip(s, 4+2+2+2+2+2+4+4+4+2+2+namelen+2+2);
    return Int(read(s, UInt64))
end
function read_jpg_from_zip(file::IO, img_num::Integer, n_features::Integer)
    archive = ZipFile.Reader(file);
    imgs = Array{RGB{N0f8}}(undef, n_features, n_features, 3, 0);
    for i in 1:img_num
        contains(archive.files[i].name, ".jpg") || continue
        @time begin
            bytes = read(archive.files[i], get_filesize(archive, i));
            img = ImageMagick.load_(bytes);
            cropped_img = crop_img(img);
            norm_img = resize_img(cropped_img, size(cropped_img)[1] / n_features);
            imgs = cat(imgs, norm_img; dims=4);
        end
    end
    return imgs
end
function customDropout(x, p::Float32, active::Bool)
    if (!active)
        return x
    end
    y = rand!(similar(x, size(x)))
    y = broadcast(el -> el>p ? 1/(1-p) : 0f0, y)
    return x .* y
end
struct LayerDesign
    kernel::Integer
    stride::Integer
    pad::Integer
end
function culc_size(design::Vector{LayerDesign}, image_size::Integer)
    size = image_size;
    for d in design
```

```
        size = (size + 2 * d.pad - d.kernel) / d.stride + 1;
    end
    return convert(Integer, size)
end
function create_discr(image_size::Integer)
    discr_design = [LayerDesign(4, 2, 1), LayerDesign(4, 2, 1)];
    return Chain(
        Conv((discr_design[1].kernel, discr_design[1].kernel), 3 => 64; stride = discr_design[1].stride, pad = discr_design[1].pad),
        x->leakyrelu.(x, 0.2f0),
        x -> customDropout(x, 0.3f0, train_mode),
        Conv((discr_design[2].kernel, discr_design[2].kernel), 64 => 128; stride = discr_design[2].stride, pad = discr_design[2].pad),
        x->leakyrelu.(x, 0.2f0),
        x -> customDropout(x, 0.3f0, train_mode),
        Flux.flatten,
        Dense(culc_size(discr_design, image_size) ^ 2 * 128, 1))
end
function create_gen(image_size::Integer)
    gen_design = [LayerDesign(4, 2, 1), LayerDesign(4, 2, 1), LayerDesign(5, 1, 2)];
    size = culc_size(gen_design, image_size);
    return Chain(Dense(latent_dim, size ^ 2 * 256),
        BatchNorm(size ^ 2 * 256, x -> leakyrelu(x, 0.2f0)),
        x -> reshape(x, size, size, 256, :),
        ConvTranspose((gen_design[3].kernel, gen_design[3].kernel), 256 => 128; stride = gen_design[3].stride, pad = gen_design[3].pad),
        BatchNorm(128, x -> leakyrelu(x, 0.2f0)),
        ConvTranspose((gen_design[2].kernel, gen_design[2].kernel), 128 => 64; stride = gen_design[2].stride, pad = gen_design[2].pad),
        BatchNorm(64, x -> leakyrelu(x, 0.2f0)),
        ConvTranspose((gen_design[1].kernel, gen_design[1].kernel), 64 => 3, tanh; stride = gen_design[1].stride, pad = gen_design[1].pad))
end

function train_dscr(discriminator, real_dataset, fake_input_data)
    all_input_data = cat(real_dataset[1], fake_input_data; dims=4);
    all_expected = hcat(real_dataset[2], zeros(1, size(fake_input_data)[end]));

    loss(input, expectedOutput) = Flux.Losses.logitbinarycrossentropy(discriminator(input), expectedOutput);
    parameters = Flux.params(discriminator);

    l, back = Zygote.pullback(() -> loss(all_input_data, all_expected), parameters);
    Flux.update!(opt_dscr, parameters, back(1.0f0));
    return l
end
function train_gen(discriminator, generator)
    noise = randn(Float32, latent_dim, batch_size) |> gpu;

    loss(input) = Flux.Losses.logitbinarycrossentropy(discriminator(generator(input)), 1.);
    parameters = Flux.params(generator);

    l, back = Zygote.pullback(() -> loss(noise), parameters);
    Flux.update!(opt_gen, parameters, back(1.0f0));
    return l
end
if s3_exists(aws, "gan-state", "dcgan.zip")
    zip_file = IOBuffer(s3_get(aws, "gan-state", "dcgan.zip"));
    dataset_size = get_file_num_in_zip(zip_file);
    batch_size = Int(round(dataset_size / 4));
    num_epochs = s3_exists(aws, "gan-state", "dcgan.json") ? JSON.parse(s3_get(aws, "gan-state", "dcgan.json"))["epoch"] : 1000;
    n_features = 100;
    latent_dim = 100;
```

```
    opt_dscr = ADAM(2e-4);
    opt_gen = ADAM(2e-4);
    train_mode = true;
    input_dataset = read_jpg_from_zip(zip_file, dataset_size, n_features)
    real_input_tensors = 2f0 * input_dataset .- 1f0;
    real_tensor_dataset = (real_input_tensors[:, :, :, 1:dataset_size], ones(1, dataset_size)) |> gpu;
    tensor_train_loader = DataLoader(real_tensor_dataset, batchsize=batch_size, shuffle=true) |> gpu
    if s3_exists(aws, "gan-state", "dcgan.bson")
        file = s3_get(aws, "gan-state", "dcgan.bson");
        gan_bson = BSON.load(IOBuffer(file));
        conv_generator = gpu(gan_bson[:generator]);
        conv_discriminator = gpu(gan_bson[:discriminator]);
    else
        conv_discriminator = create_discr(n_features) |> gpu;
        conv_generator = create_gen(n_features) |> gpu;
    end
    errors_dscr = Array{Float32}(undef, 0, 2);
    errors_gen = Array{Float32}(undef, 0, 2);
    for epoch in 1:num_epochs
        @time begin
            l_dscr = Array{Float32}(undef, 0);
            l_gen = Array{Float32}(undef, 0);
            for (x,y) in tensor_train_loader
                noise = randn(Float32, latent_dim, batch_size) |> gpu;
                fake_input_data = conv_generator(noise);
                l_dscr = vcat(l_dscr, train_dscr(conv_discriminator, (x, y), fake_input_data));
                l_gen = vcat(l_gen, train_gen(conv_discriminator, conv_generator));
            end
            err_dscr = sum(l_dscr) / size(l_dscr)[1];
            err_gen = sum(l_gen) / size(l_gen)[1];
            global errors_dscr = vcat(errors_dscr, [size(errors_dscr)[1]+1 err_dscr]);
            global errors_gen = vcat(errors_gen, [size(errors_gen)[1]+1 err_gen]);
        end
    end
    train_mode = false;
    bytes = Array{UInt8}(undef, 0);
    buf = IOBuffer(bytes, read=true, write=true);
    BSON.bson(buf, Dict(:generator => cpu(conv_generator), :discriminator => cpu(conv_discriminator)));
    s3_put(aws, "gan-state", "dcgan.bson", bytes);
    n = randn(Float32, 100, 9) |> gpu;
    a = 0.5f0 * cpu(conv_generator(n)) .+ 0.5f0;
    to_save_1 = cat(a[:, :, :, 1], a[:, :, :, 2], a[:, :, :, 3]; dims=2);
    to_save_2 = cat(a[:, :, :, 4], a[:, :, :, 5], a[:, :, :, 6]; dims=2);
    to_save_3 = cat(a[:, :, :, 7], a[:, :, :, 8], a[:, :, :, 9]; dims=2);
    to_save = cat(to_save_1, to_save_2, to_save_3; dims=1);
    img = colorview(RGB, permutedims([N0f8.(to_save[:, :, 1]);;; N0f8.(to_save[:, :, 2]);;; N0f8.(to_save[:, :, 3])],
(3,1,2)));
    s3_put(aws, "gan-state", "dcgan.jpg", getblob(image2wand(img), "JPEG"));
end
```

A.2. Код модуля керування тренувальними наборами

```
package org.ai.machineart.dataset;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class DataSetManagerApplication {
        public static void main(String[] args) {
                SpringApplication.run(DataSetManagerApplication.class, args);
        }
}
```

```
package org.ai.machineart.model;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBAutoGeneratedKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBHashKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBIgnore;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBTable;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.experimental.SuperBuilder;
import java.util.Date;
@Data
@AllArgsConstructor
@NoArgsConstructor
@SuperBuilder
@DynamoDBTable(tableName = DataSet.TABLE_NAME)
public class DataSet {
   @DynamoDBIgnore
   public static final String TABLE_NAME = "DataSet";
   @DynamoDBHashKey
   @DynamoDBAutoGeneratedKey
   private String id;
   private String tag;
   private Date date;
   private String fileName;
   private long fileSize;
}

package org.ai.machineart.dto;
import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.experimental.SuperBuilder;
import java.util.Date;
@Data
@SuperBuilder
@AllArgsConstructor
@NoArgsConstructor
@JsonInclude(JsonInclude.Include.NON_NULL)
public class DataSetDto {
   private String id;
   private String tag;
   private Date date;
   private PictureMetadataDto pictureMetadata;
}

package org.ai.machineart.dto;
import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.experimental.SuperBuilder;
@Data
@SuperBuilder
@AllArgsConstructor
@NoArgsConstructor
@JsonInclude(JsonInclude.Include.NON_NULL)
public class DataSetPictureDto {
   private String id;
   private String fileName;
   private byte[] picturesFile;
}
```

```
package org.ai.machineart.dto;
import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.experimental.SuperBuilder;
@Data
@SuperBuilder
@AllArgsConstructor
@NoArgsConstructor
@JsonInclude(JsonInclude.Include.NON_NULL)
public class PictureMetadataDto {
   private String dataSetId;
   private String fileName;
   private String fileType;
   private long fileSize;
}

package org.ai.machineart.dto;
import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.experimental.SuperBuilder;
@Data
@SuperBuilder
@NoArgsConstructor
@AllArgsConstructor
@JsonInclude(JsonInclude.Include.NON_NULL)
public class ErrorDto {
   private String errorName;
   private String value;
}

package org.ai.machineart.utils;
import com.amazonaws.services.s3.model.S3Object;
import java.io.IOException;
public class ByteArrayConverter {
   private ByteArrayConverter() {}
   public static byte[] getByteArrayFromS3Object(S3Object s3Object) {
      try {
         return s3Object.getObjectContent().readAllBytes();
      } catch (IOException e) {
         throw new RuntimeException(e);
      }
   }
}

package org.ai.machineart.dataset.config;
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClient;
import org.socialsignin.spring.data.dynamodb.repository.config.EnableDynamoDBRepositories;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
@Configuration
@EnableDynamoDBRepositories(basePackages = "org.ai.machineart.dataset.repository")
public class DynamoDBConfig {
   @Value("${amazon.dynamodb.endpoint}")
   private String endpoint;
```

```
    @Value("${amazon.aws.region}")
    private String region;
    @Bean
    public AmazonDynamoDB amazonDynamoDB() {
        return AmazonDynamoDBClient.builder()
                .withCredentials(amazonAWSCredentials())
                .withEndpointConfiguration(new AwsClientBuilder.EndpointConfiguration(endpoint, region))
                .build();
    }
    @Bean
    public AWSCredentialsProvider amazonAWSCredentials() {
        return new ProfileCredentialsProvider("default");
    }
}

package org.ai.machineart.dataset.config;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBMapper;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBTable;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
import org.reflections.Reflections;
import org.springframework.stereotype.Component;
import java.lang.reflect.Field;
import java.util.Arrays;
import java.util.List;
import java.util.Set;
import static org.reflections.scanners.Scanners.SubTypes;
import static org.reflections.scanners.Scanners.TypesAnnotated;
@Component
public class DynamoDBTablesCreator {
    private static final String TABLE_NAME_FIELD_NAME = "TABLE_NAME";
    private static final String MODELS_PACKAGE = "org.ai.machineart.model";
    private final DynamoDBMapper dynamoDBMapper;
    private final AmazonDynamoDB db;
    public DynamoDBTablesCreator(DynamoDBMapper dynamoDBMapper, AmazonDynamoDB db) {
        this.dynamoDBMapper = dynamoDBMapper;
        this.db = db;
        initTables();
    }
    public void initTables() {
        List<String> existingTables = db.listTables().getTableNames();
        Reflections reflections = new Reflections(MODELS_PACKAGE);
        Set<Class<?>> models = reflections.get(SubTypes.of(TypesAnnotated.with(DynamoDBTable.class)).asClass());
        models.stream()
                .filter(el -> Arrays.stream(el.getFields())
                        .map(Field::getName)
                        .toList()
                        .contains(TABLE_NAME_FIELD_NAME))
                .filter(el -> !existingTables.contains(getTableNameFieldValue(el)))
                .forEach(this::createTable);
    }
    private void createTable(Class<?> model) {
        CreateTableRequest dataSetCreateRequest = dynamoDBMapper.generateCreateTableRequest(model);
        dataSetCreateRequest.setProvisionedThroughput(new ProvisionedThroughput(1L, 1L));
        db.createTable(dataSetCreateRequest);
    }
    private String getTableNameFieldValue(Class<?> clazz) {
        try {
            return (String) clazz.getField(TABLE_NAME_FIELD_NAME).get(clazz);
        } catch (IllegalAccessException | NoSuchFieldException e) {
            throw new RuntimeException(e);
        }
```

```
    }
}

package org.ai.machineart.dataset.config;
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
@Configuration
public class S3Config {
    @Value("${amazon.aws.region}")
    private String region;
    @Bean
    public AmazonS3 amazonS3(AWSCredentialsProvider amazonAWSCredentials) {
        return AmazonS3Client.builder()
                .withCredentials(amazonAWSCredentials)
                .withRegion(region)
                .build();
    }
}

package org.ai.machineart.dataset.config;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
@Configuration
public class SwaggerConfig {
    @Bean
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2)
                .select()
                .apis(RequestHandlerSelectors.basePackage("org.ai.machineart.dataset.controller"))
                .paths(PathSelectors.any())
                .build();
    }
}

package org.ai.machineart.dataset.controller;
import io.swagger.annotations.Api;
import lombok.AllArgsConstructor;
import org.ai.machineart.dataset.dto.CreateResponse;
import org.ai.machineart.dataset.dto.DataSetMetadataDto;
import org.ai.machineart.dataset.exception.WrongContentTypeException;
import org.ai.machineart.dataset.facade.DataSetFacade;
import org.ai.machineart.dto.DataSetDto;
import org.ai.machineart.dto.DataSetPictureDto;
import org.springframework.core.io.InputStreamResource;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.net.URLConnection;
import java.util.List;
import java.util.Objects;
```

```
import java.util.stream.Collectors;
@RestController
@AllArgsConstructor
@RequestMapping("/dataset")
@Api(tags = "Data Set")
public class DataSetController {
    private final DataSetFacade dataSetFacade;
    @GetMapping("/getAll")
    public ResponseEntity<List<String>> gatAllDataSetIds() {
        List<String> dataSetIds = dataSetFacade.findAll().stream()
                .map(DataSetDto::getId)
                .collect(Collectors.toList());
        return new ResponseEntity<>(dataSetIds, HttpStatus.OK);
    }
    @GetMapping("/getMetadata/{id}")
    public ResponseEntity<DataSetDto> gatDataSetById(@PathVariable String id) {
        return new ResponseEntity<>(dataSetFacade.findById(id), HttpStatus.OK);
    }
    @GetMapping("/getPictureFile/{id}")
    public ResponseEntity<InputStreamResource> gatPictureFileById(@PathVariable String id) {
        DataSetPictureDto picture = dataSetFacade.findPictureById(id);
        String contentType = URLConnection.guessContentTypeFromName(picture.getFileName());
        HttpHeaders headers = new HttpHeaders();
        headers.add("Content-Disposition", "attachment;filename=" + picture.getFileName());
        return ResponseEntity.ok()
                .headers(headers)
                .contentLength(picture.getPicturesFile().length)
                .contentType(MediaType.parseMediaType(contentType))
                .body(new InputStreamResource(new ByteArrayInputStream(picture.getPicturesFile())));
    }
    @PostMapping(value = "/saveMetadata")
    public ResponseEntity<CreateResponse> createDataSetEntity(@RequestBody DataSetMetadataDto dto) {
        return new ResponseEntity<>(dataSetFacade.saveMetadata(dto), HttpStatus.CREATED);
    }
    @PutMapping(value = "/savePicture/{id}")
    public ResponseEntity<CreateResponse> createDataSetEntity(@PathVariable String id,
                                                @RequestPart(name = "picture") MultipartFile file) throws IOException {
        if (!isSupportedContentType(file.getContentType())) {
            throw new WrongContentTypeException("Content-type [" + file.getContentType() + "] is not supported");
        }
        DataSetPictureDto pictureDto = DataSetPictureDto.builder()
                .id(id)
                .fileName(id + getFileFormatFromContentType(Objects.requireNonNull(file.getContentType())))
                .picturesFile(file.getBytes())
                .build();
        return new ResponseEntity<>(dataSetFacade.savePicture(pictureDto), HttpStatus.OK);
    }
    @DeleteMapping("/delete/{id}")
    public ResponseEntity<Void> deleteDataSetEntity(@PathVariable String id) {
        dataSetFacade.deleteById(id);
        return ResponseEntity.ok().build();
    }
    private boolean isSupportedContentType(String contentType) {
        return Objects.nonNull(contentType) && contentType.equals("application/zip");
    }
    private String getFileFormatFromContentType(String contentType) {
        return "." + contentType.split("[/]")[1];
    }
}

package org.ai.machineart.dataset.dto;
import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.AllArgsConstructor;
```

```
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.experimental.SuperBuilder;
@Data
@SuperBuilder
@AllArgsConstructor
@NoArgsConstructor
@JsonInclude(JsonInclude.Include.NON_NULL)
public class CreateResponse {
   private String dataSetId;
}

package org.ai.machineart.dataset.dto;
import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.experimental.SuperBuilder;
import java.util.Date;
@Data
@SuperBuilder
@AllArgsConstructor
@NoArgsConstructor
@JsonInclude(JsonInclude.Include.NON_NULL)
public class DataSetMetadataDto {
   private String tag;
   private Date date;
}

package org.ai.machineart.dataset.facade.impl;
import lombok.RequiredArgsConstructor;
import org.ai.machineart.dataset.dto.CreateResponse;
import org.ai.machineart.dataset.dto.DataSetMetadataDto;
import org.ai.machineart.dataset.exception.PictureNotFoundException;
import org.ai.machineart.dataset.facade.DataSetFacade;
import org.ai.machineart.dataset.mapper.DataSetMapper;
import org.ai.machineart.dataset.service.DataSetService;
import org.ai.machineart.dto.DataSetDto;
import org.ai.machineart.dto.DataSetPictureDto;
import org.ai.machineart.model.DataSet;
import org.springframework.stereotype.Service;
import java.util.List;
import java.util.stream.Collectors;
@Service
@RequiredArgsConstructor
public class DataSetFacadeImpl implements DataSetFacade {
   private final DataSetService dataSetService;
   private final DataSetMapper dataSetMapper;
   @Override
   public CreateResponse saveMetadata(DataSetMetadataDto dataSetDto) {
      DataSet createdDataSet = dataSetService.save(dataSetMapper.toDao(dataSetDto));
      return CreateResponse.builder()
            .dataSetId(createdDataSet.getId())
            .build();
   }
   @Override
   public CreateResponse savePicture(DataSetPictureDto dataSetPictureDto) {
      DataSet dataSet = dataSetService.findById(dataSetPictureDto.getId());
      dataSet.setFileName(dataSetPictureDto.getFileName());
      dataSet.setFileSize(dataSetPictureDto.getPicturesFile().length);
      dataSet = dataSetService.update(dataSet);
      dataSetService.saveToS3(dataSet, dataSetPictureDto.getPicturesFile());
      return CreateResponse.builder()
```

```
                .dataSetId(dataSet.getId())
                .build();
    }
    @Override
    public DataSetDto findById(String id) {
        return dataSetMapper.toDto(dataSetService.findById(id));
    }
    @Override
    public List<DataSetDto> findByTag(String tag) {
        return dataSetService.findByTag(tag).stream()
                .map(dataSetMapper::toDto)
                .collect(Collectors.toList());
    }
    @Override
    public List<DataSetDto> findAll() {
        return dataSetService.findAll().stream()
                .map(dataSetMapper::toDto)
                .collect(Collectors.toList());
    }
    @Override
    public DataSetPictureDto findPictureById(String id) {
        DataSet dataSet = dataSetService.findById(id);
        if (dataSet.getFileSize() == 0) {
            throw new PictureNotFoundException("There is no uploaded pictures for data set [" + id + "]");
        }
        return dataSetMapper.toPictureDto(dataSet, dataSetService.findPicturesFileByFilename(dataSet.getFileName()));
    }
    @Override
    public void deleteById(String id) {
        dataSetService.deleteById(id);
    }
    @Override
    public void deleteAll() {
        dataSetService.deleteAll();
    }
}

package org.ai.machineart.dataset.handler;
import lombok.extern.slf4j.Slf4j;
import org.ai.machineart.exception.DataSetNotFoundException;
import org.ai.machineart.dataset.exception.PictureNotFoundException;
import org.ai.machineart.dataset.exception.WrongContentTypeException;
import org.ai.machineart.dto.ErrorDto;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
@Slf4j
@ControllerAdvice
public class GlobalExceptionHandler {
    @ExceptionHandler({DataSetNotFoundException.class, PictureNotFoundException.class})
    public final ResponseEntity<ErrorDto> handleNotFoundException(Exception ex) {
        log.warn(ex.getMessage());
        return new ResponseEntity<>(new ErrorDto(ex.getClass().getSimpleName(), ex.getMessage()),
HttpStatus.NOT_FOUND);
    }
    @ExceptionHandler({WrongContentTypeException.class})
    public final ResponseEntity<ErrorDto> handleDuplicateException(Exception ex) {
        log.warn(ex.getMessage());
        return new ResponseEntity<>(new ErrorDto(ex.getClass().getSimpleName(), ex.getMessage()),
HttpStatus.CONFLICT);
    }
}
```

```
package org.ai.machineart.dataset.mapper;
import org.ai.machineart.dataset.dto.DataSetMetadataDto;
import org.ai.machineart.dto.DataSetDto;
import org.ai.machineart.dto.DataSetPictureDto;
import org.ai.machineart.dto.PictureMetadataDto;
import org.ai.machineart.model.DataSet;
import org.springframework.stereotype.Component;
import java.net.URLConnection;
import java.util.Objects;
@Component
public class DataSetMapper {
   public DataSetDto toDto(DataSet dataSet) {
      String fileType = Objects.isNull(dataSet.getFileName()) || dataSet.getFileName().isBlank()
            ? ""
            : URLConnection.guessContentTypeFromName(dataSet.getFileName());
      PictureMetadataDto pictureMetadata = PictureMetadataDto.builder()
            .dataSetId(dataSet.getId())
            .fileName(dataSet.getFileName())
            .fileSize(dataSet.getFileSize())
            .fileType(fileType)
            .build();
      return DataSetDto.builder()
            .id(dataSet.getId())
            .tag(dataSet.getTag())
            .date(dataSet.getDate())
            .pictureMetadata(pictureMetadata)
            .build();
   }
   public DataSetPictureDto toPictureDto(DataSet dataSet, byte[] pictureFile) {
      return DataSetPictureDto.builder()
            .id(dataSet.getId())
            .fileName(dataSet.getFileName())
            .picturesFile(pictureFile)
            .build();
   }
   public DataSet toDao(DataSetMetadataDto dataSetDto) {
      return DataSet.builder()
            .tag(dataSetDto.getTag())
            .date(dataSetDto.getDate())
            .fileName("")
            .fileSize(0)
            .build();
   }
}

package org.ai.machineart.dataset.repository;
import org.ai.machineart.model.DataSet;
import org.socialsignin.spring.data.dynamodb.repository.EnableScan;
import org.springframework.data.repository.CrudRepository;
import java.util.List;
@EnableScan
public interface DataSetRepository extends CrudRepository<DataSet, String> {
   List<DataSet> findByTag(String tag);
}

package org.ai.machineart.dataset.service.impl;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.model.ObjectMetadata;
import lombok.RequiredArgsConstructor;
import org.ai.machineart.dataset.repository.DataSetRepository;
import org.ai.machineart.dataset.service.DataSetService;
import org.ai.machineart.exception.DataSetNotFoundException;
```

```
import org.ai.machineart.model.DataSet;
import org.ai.machineart.utils.ByteArrayConverter;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import java.io.ByteArrayInputStream;
import java.util.List;
import java.util.Objects;
import java.util.stream.Collectors;
import java.util.stream.StreamSupport;
@Service
@RequiredArgsConstructor
public class DataSetServiceImpl implements DataSetService {
   @Value("${amazon.s3.bucket.name.dataset}")
   private String bucketName;
   private final DataSetRepository dataSetRepository;
   private final AmazonS3 amazonS3;
   @Override
   public DataSet save(DataSet dataSet) {
      return dataSetRepository.save(dataSet);
   }
   @Override
   public String saveToS3(DataSet dataSet, byte[] picturesFile) {
      amazonS3.putObject(bucketName, dataSet.getFileName(), new ByteArrayInputStream(picturesFile), new
ObjectMetadata());
      return dataSet.getFileName();
   }
   @Override
   public DataSet update(DataSet dataSet) {
      String id = dataSet.getId();
      if (Objects.isNull(id) || id.isBlank() || dataSetRepository.findById(id).isEmpty()) {
         throw new DataSetNotFoundException(getNotFoundMessageForId(id));
      }
      return dataSetRepository.save(dataSet);
   }
   @Override
   public DataSet findById(String id) {
      if (Objects.isNull(id) || id.isBlank()) {
         throw new DataSetNotFoundException(getNotFoundMessageForId(id));
      }
      return dataSetRepository.findById(id)
            .orElseThrow(() -> new DataSetNotFoundException(getNotFoundMessageForId(id)));
   }
   @Override
   public byte[] findPicturesFileByFilename(String filename) {
      if (!amazonS3.doesObjectExist(bucketName, filename)) {
         throw new DataSetNotFoundException("Pictures file " + filename + " not found");
      }
      return ByteArrayConverter.getByteArrayFromS3Object(amazonS3.getObject(bucketName, filename));
   }
   @Override
   public List<DataSet> findByTag(String tag) {
      if (Objects.isNull(tag) || tag.isBlank()) {
         throw new DataSetNotFoundException(getNotFoundMessageForTag(tag));
      }
      List<DataSet> dataSets = dataSetRepository.findByTag(tag);
      if (dataSets.isEmpty()) {
         throw new DataSetNotFoundException(getNotFoundMessageForTag(tag));
      }
      return dataSets;
   }
   @Override
   public List<DataSet> findAll() {
      return StreamSupport.stream(dataSetRepository.findAll().spliterator(), false)
```

```
            .collect(Collectors.toList());
    }
    @Override
    public void deleteById(String id) {
        DataSet dataSet = findById(id);
        amazonS3.deleteObject(bucketName, dataSet.getFileName());
        dataSetRepository.deleteById(id);
    }
    @Override
    public void deleteAll() {
        findAll().forEach(el -> amazonS3.deleteObject(bucketName, el.getFileName()));
        dataSetRepository.deleteAll();
    }
    private String getNotFoundMessageForId(String id) {
        if (Objects.isNull(id) || id.isBlank()) {
            return "Please provide dataSetId";
        }
        return "Data set with id " + id + " not found";
    }
    private String getNotFoundMessageForTag(String tag) {
        if (Objects.isNull(tag) || tag.isBlank()) {
            return "Please provide tag of arts";
        }
        return "Data sets of tag " + tag + " not found";
    }
}
```

А.3. Код модуля керування станом моделі

```
package org.ai.machineart.gan;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class GanManagerApplication {
            public static void main(String[] args) {
                        SpringApplication.run(GanManagerApplication.class, args);
            }
}

package org.ai.machineart.gan.config;
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import software.amazon.awssdk.auth.credentials.AwsCredentialsProvider;
@Configuration
public class CredentialsConfig {
    @Bean
    public AWSCredentialsProvider amazonAWSCredentials() {
        return new ProfileCredentialsProvider("default");
    }
    @Bean
    public AwsCredentialsProvider amazonAWSCredentialsV2() {
        return software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider.builder()
                .profileName("default")
                .build();
    }
}

package org.ai.machineart.gan.config;
import com.amazonaws.auth.AWSCredentialsProvider;
```

```
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClient;
import org.socialsignin.spring.data.dynamodb.repository.config.EnableDynamoDBRepositories;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.scheduling.annotation.EnableScheduling;
@Configuration
@EnableDynamoDBRepositories(basePackages = "org.ai.machineart.gan.repository")
@EnableScheduling
public class DynamoDBConfig {
   @Value("${amazon.dynamodb.endpoint}")
   private String endpoint;
   @Value("${amazon.aws.region}")
   private String region;
   @Bean
   public AmazonDynamoDB amazonDynamoDB(AWSCredentialsProvider amazonAWSCredentials) {
      return AmazonDynamoDBClient.builder()
            .withCredentials(amazonAWSCredentials)
            .withEndpointConfiguration(new AwsClientBuilder.EndpointConfiguration(endpoint, region))
            .build();
   }
}

package org.ai.machineart.gan.config;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBMapper;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBTable;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
import org.reflections.Reflections;
import org.springframework.stereotype.Component;
import java.lang.reflect.Field;
import java.util.Arrays;
import java.util.List;
import java.util.Set;
import static org.reflections.scanners.Scanners.SubTypes;
import static org.reflections.scanners.Scanners.TypesAnnotated;
@Component
public class DynamoDBTablesCreator {
   private static final String TABLE_NAME_FIELD_NAME = "TABLE_NAME";
   private static final String MODELS_PACKAGE = "org.ai.machineart.gan.model";
   private final DynamoDBMapper dynamoDBMapper;
   private final AmazonDynamoDB db;
   public DynamoDBTablesCreator(DynamoDBMapper dynamoDBMapper, AmazonDynamoDB db) {
      this.dynamoDBMapper = dynamoDBMapper;
      this.db = db;
      initTables();
   }
   public void initTables() {
      List<String> existingTables = db.listTables().getTableNames();
      Reflections reflections = new Reflections(MODELS_PACKAGE);
      Set<Class<?>> models = reflections.get(SubTypes.of(TypesAnnotated.with(DynamoDBTable.class)).asClass());
      models.stream()
            .filter(el -> Arrays.stream(el.getFields())
                  .map(Field::getName)
                  .toList()
                  .contains(TABLE_NAME_FIELD_NAME))
            .filter(el -> !existingTables.contains(getTableNameFieldValue(el)))
            .forEach(this::createTable);
   }
   private void createTable(Class<?> model) {
```

```
        CreateTableRequest dataSetCreateRequest = dynamoDBMapper.generateCreateTableRequest(model);
        dataSetCreateRequest.setProvisionedThroughput(new ProvisionedThroughput(1L, 1L));
        db.createTable(dataSetCreateRequest);
    }
    private String getTableNameFieldValue(Class<?> clazz) {
        try {
            return (String) clazz.getField(TABLE_NAME_FIELD_NAME).get(clazz);
        } catch (IllegalAccessException | NoSuchFieldException e) {
            throw new RuntimeException(e);
        }
    }
}
```

```
package org.ai.machineart.gan.config;
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
@Configuration
public class S3Config {
    @Value("${amazon.aws.region}")
    private String region;
    @Bean
    public AmazonS3 amazonS3(AWSCredentialsProvider amazonAWSCredentials) {
        return AmazonS3Client.builder()
                .withCredentials(amazonAWSCredentials)
                .withRegion(region)
                .build();
    }
}
```

```
package org.ai.machineart.gan.config;
import org.ai.machineart.gan.service.SageMakerService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.Resource;
import org.springframework.core.io.ResourceLoader;
import org.springframework.util.StreamUtils;
import software.amazon.awssdk.auth.credentials.AwsCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sagemaker.SageMakerClient;
import java.io.IOException;
import java.io.InputStream;
import java.nio.charset.Charset;
import java.util.Arrays;
import java.util.Map;
import java.util.stream.Collectors;
@Configuration
public class SageMakerConfig {
    @Value("${amazon.aws.region}")
    private String region;
    @Autowired
    private ResourceLoader resourceLoader;
    @Bean
    public SageMakerClient sageMakerClient(AwsCredentialsProvider amazonAWSCredentialsV2) {
        return SageMakerClient.builder()
                .credentialsProvider(amazonAWSCredentialsV2)
                .region(Region.of(region))
                .build();
```

```
    }
    @Bean
    public Map<SageMakerService.ConfigType, String> resourceConfigScripts() {
        return Arrays.stream(SageMakerService.ConfigType.values())
                .map(el -> Map.entry(el, resourceLoader.getResource("classpath:" + el.getConfigFileName())))
                .collect(Collectors.toMap(Map.Entry::getKey, el -> convertToString(el.getValue())));
    }
    private String convertToString(Resource r) {
        try (InputStream is = r.getInputStream()) {
            return StreamUtils.copyToString(is, Charset.defaultCharset());
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
}
```

```
package org.ai.machineart.gan.config;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
@Configuration
public class SwaggerConfig {
    @Bean
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2)
                .select()
                .apis(RequestHandlerSelectors.basePackage("org.ai.machineart.gan.controller"))
                .paths(PathSelectors.any())
                .build();
    }
}
```

```
package org.ai.machineart.gan.controller;
import io.swagger.annotations.Api;
import lombok.AllArgsConstructor;
import org.ai.machineart.gan.dto.CreateResponse;
import org.ai.machineart.gan.dto.GanTestRequest;
import org.ai.machineart.gan.dto.GanTrainRequest;
import org.ai.machineart.gan.dto.StateDto;
import org.ai.machineart.gan.facade.StateFacade;
import org.springframework.core.io.InputStreamResource;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.io.ByteArrayInputStream;
import java.util.List;
@RestController
@AllArgsConstructor
@RequestMapping("/gan")
@Api(tags = "GAN")
public class StateController {
    private final StateFacade stateFacade;
    @GetMapping("/getAll")
    public ResponseEntity<List<StateDto>> gatAllStates() {
        return new ResponseEntity<>(stateFacade.findAll(), HttpStatus.OK);
    }
    @GetMapping("/getGenerated/{id}")
    public ResponseEntity<InputStreamResource> gatPictures(@PathVariable String id) {
```

```
        ByteArrayInputStream bais = stateFacade.findZippedGeneratedPicturesById(id);
        HttpHeaders headers = new HttpHeaders();
        headers.add("Content-Disposition", "attachment;filename=pictures.zip");
        return ResponseEntity.ok()
                .headers(headers)
                .contentType(MediaType.parseMediaType("application/zip"))
                .body(new InputStreamResource(bais));
    }
    @PostMapping("/test/{id}")
    public ResponseEntity<Void> testGenerator(@PathVariable String id) {
        stateFacade.testGan(GanTestRequest.builder().stateId(id).build());
        return ResponseEntity.status(HttpStatus.ACCEPTED).build();
    }
    @PostMapping("/train/{datasetId}/{epochs}")
    public ResponseEntity<CreateResponse> train(@PathVariable String datasetId, @PathVariable int epochs) {
        GanTrainRequest request = GanTrainRequest.builder()
                .datasetId(datasetId)
                .epochs(epochs)
                .continued(false)
                .build();
        String stateId = stateFacade.trainGan(request);
        return ResponseEntity.status(HttpStatus.CREATED).body(CreateResponse.builder()
                .stateId(stateId)
                .build());
    }
    @PutMapping("/train/continued/{stateId}/{epochs}")
    public ResponseEntity<Void> continueTraining(@PathVariable String stateId, @PathVariable int epochs) {
        GanTrainRequest request = GanTrainRequest.builder()
                .stateId(stateId)
                .epochs(epochs)
                .continued(true)
                .build();
        stateFacade.trainGan(request);
        return ResponseEntity.status(HttpStatus.ACCEPTED).build();
    }
    @DeleteMapping("/delete/{id}")
    public ResponseEntity<Void> deleteStateEntity(@PathVariable String id) {
        stateFacade.deleteById(id);
        return ResponseEntity.ok().build();
    }
}

package org.ai.machineart.gan.dto;
import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.experimental.SuperBuilder;
@Data
@SuperBuilder
@AllArgsConstructor
@NoArgsConstructor
@JsonInclude(JsonInclude.Include.NON_NULL)
public class CreateResponse {
    private String stateId;
}

package org.ai.machineart.gan.dto;
import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.experimental.SuperBuilder;
```

```
@Data
@SuperBuilder
@AllArgsConstructor
@NoArgsConstructor
@JsonInclude(JsonInclude.Include.NON_NULL)
public class GanTestRequest {
   private String stateId;
}

package org.ai.machineart.gan.dto;
import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.experimental.SuperBuilder;
@Data
@SuperBuilder
@AllArgsConstructor
@NoArgsConstructor
@JsonInclude(JsonInclude.Include.NON_NULL)
public class GanTrainRequest {
   private String stateId;
   private String datasetId;
   private int epochs;
   private boolean continued;
}

package org.ai.machineart.gan.dto;
import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.experimental.SuperBuilder;
@Data
@SuperBuilder
@AllArgsConstructor
@NoArgsConstructor
@JsonInclude(JsonInclude.Include.NON_NULL)
public class StateDto {
   private String stateId;
   private String generatorAccuracy;
   private String discriminatorAccuracy;
   private String tag;
}

package org.ai.machineart.gan.dto;
import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.experimental.SuperBuilder;
@Data
@SuperBuilder
@AllArgsConstructor
@NoArgsConstructor
@JsonInclude(JsonInclude.Include.NON_NULL)
public class StateCreateDto {
   private String datasetId;
   private String tag;
   private byte[] bsonFile;
}

package org.ai.machineart.gan.facade.impl;
```

```
import lombok.RequiredArgsConstructor;
import org.ai.machineart.gan.dto.GanTestRequest;
import org.ai.machineart.gan.dto.GanTrainRequest;
import org.ai.machineart.gan.dto.StateCreateDto;
import org.ai.machineart.gan.dto.StateDto;
import org.ai.machineart.gan.exception.MoreThanOneActiveStateException;
import org.ai.machineart.gan.facade.StateFacade;
import org.ai.machineart.gan.mapper.StateMapper;
import org.ai.machineart.gan.model.State;
import org.ai.machineart.gan.service.DataSetService;
import org.ai.machineart.gan.service.SageMakerService;
import org.ai.machineart.gan.service.SageMakerService.ConfigType;
import org.ai.machineart.gan.service.StateService;
import org.ai.machineart.gan.service.ZipService;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Service;
import java.io.ByteArrayInputStream;
import java.util.List;
import java.util.Map;
import java.util.Objects;
import java.util.stream.Collectors;
@Service
@RequiredArgsConstructor
public class StateFacadeImpl implements StateFacade {
    private final StateService stateService;
    private final ZipService zipService;
    private final SageMakerService sageMakerService;
    private final DataSetService dataSetService;
    private final StateMapper stateMapper;
    @Override
    public String save(StateCreateDto stateCreateDto) {
        State state = stateMapper.toDao(stateCreateDto);
        stateService.saveToS3(state, stateCreateDto.getBsonFile());
        stateService.save(state);
        return state.getStateFileName();
    }
    @Override
    public void testGan(GanTestRequest request) {
        if (checkStateActivity()) {
            throw new MoreThanOneActiveStateException("Application can have only one active state at the same time");
        }
        State state = stateService.findById(request.getStateId());
        stateService.activateTesting(state);
        sageMakerService.loadStartConfig(ConfigType.TEST);
        sageMakerService.startInstance();
    }
    @Override
    public String trainGan(GanTrainRequest request) {
        if (checkStateActivity()) {
            throw new MoreThanOneActiveStateException("Application can have only one active state at the same time");
        }
        State state;
        if (request.isContinued() && Objects.nonNull(request.getStateId()) && !request.getStateId().isBlank()) {
            state = stateService.findById(request.getStateId());
        } else {
            String tag = dataSetService.findById(request.getDatasetId()).getTag();
            state = stateMapper.emptyDao(tag, request.getDatasetId());
            stateService.save(state);
        }
        byte[] zipFile = dataSetService.findZipFileById(state.getDatasetId());
        stateService.activateTraining(state, zipFile, request.getEpochs());
        sageMakerService.loadStartConfig(ConfigType.TRAIN);
        sageMakerService.startInstance();
```

```
        return state.getId();
    }
    @Scheduled(fixedDelay = 60000, initialDelay = 60000)
    @Override
    public boolean checkStateActivity() {
        List<State> activeStates = stateService.findAll().stream()
                .filter(State::isActive)
                .toList();
        if (activeStates.size() > 1) {
            throw new MoreThanOneActiveStateException("Application can have only one active state at the same time");
        } else if (activeStates.size() == 0) {
            return false;
        }
        if (!stateService.deactivate(activeStates.get(0))) {
            return true;
        }
        sageMakerService.stopInstance();
        sageMakerService.loadStartConfig(ConfigType.IDLE);
        return false;
    }
    @Override
    public StateDto findById(String id) {
        return stateMapper.toDto(stateService.findById(id));
    }
    @Override
    public ByteArrayInputStream findZippedGeneratedPicturesById(String id) {
        Map<String, byte[]> pictures = stateService.findGeneratedPicturesById(id);
        return zipService.zipToSingleByteArray(pictures);
    }
    @Override
    public List<StateDto> findByTag(String tag) {
        return stateService.findByTag(tag).stream()
                .map(stateMapper::toDto)
                .collect(Collectors.toList());
    }
    @Override
    public List<StateDto> findAll() {
        return stateService.findAll().stream()
                .map(stateMapper::toDto)
                .collect(Collectors.toList());
    }
    @Override
    public void deleteById(String id) {
        stateService.deleteById(id);
    }
}

package org.ai.machineart.gan.handler;
import lombok.extern.slf4j.Slf4j;
import org.ai.machineart.dto.ErrorDto;
import org.ai.machineart.exception.DataSetNotFoundException;
import org.ai.machineart.exception.StateNotFoundException;
import org.ai.machineart.gan.exception.MoreThanOneActiveStateException;
import org.ai.machineart.gan.exception.WrongContentTypeException;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
@Slf4j
@ControllerAdvice
public class GlobalExceptionHandler {
    @ExceptionHandler({DataSetNotFoundException.class, StateNotFoundException.class})
    public final ResponseEntity<ErrorDto> handleNotFoundException(Exception ex) {
```

```
        log.warn(ex.getMessage());
        return new ResponseEntity<>(new ErrorDto(ex.getClass().getSimpleName(), ex.getMessage()),
HttpStatus.NOT_FOUND);
    }
    @ExceptionHandler({WrongContentTypeException.class, MoreThanOneActiveStateException.class})
    public final ResponseEntity<ErrorDto> handleConflictException(Exception ex) {
        log.warn(ex.getMessage());
        return new ResponseEntity<>(new ErrorDto(ex.getClass().getSimpleName(), ex.getMessage()),
HttpStatus.CONFLICT);
    }
}

package org.ai.machineart.gan.mapper;
import org.ai.machineart.gan.dto.StateCreateDto;
import org.ai.machineart.gan.dto.StateDto;
import org.ai.machineart.gan.model.State;
import org.springframework.stereotype.Component;
import java.util.UUID;
@Component
public class StateMapper {
    public StateDto toDto(State state) {
        return StateDto.builder()
                .stateId(state.getId())
                .generatorAccuracy(state.getGeneratorAccuracy())
                .discriminatorAccuracy(state.getDiscriminatorAccuracy())
                .tag(state.getTag())
                .build();
    }
    public State toDao(StateCreateDto createDto) {
        String id = UUID.randomUUID().toString();
        return State.builder()
                .id(id)
                .stateFileName(id + ".bson")
                .datasetId(createDto.getDatasetId())
                .tag(createDto.getTag())
                .generatorAccuracy("")
                .discriminatorAccuracy("")
                .active(false)
                .build();
    }
    public State emptyDao(String tag, String datasetId) {
        String id = UUID.randomUUID().toString();
        return State.builder()
                .id(id)
                .stateFileName(id + ".bson")
                .datasetId(datasetId)
                .tag(tag)
                .generatorAccuracy("")
                .discriminatorAccuracy("")
                .active(false)
                .build();
    }
}

package org.ai.machineart.gan.model;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBHashKey;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBIgnore;
import com.amazonaws.services.dynamodbv2.datamodeling.DynamoDBTable;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.experimental.SuperBuilder;
@Data
```

```
@SuperBuilder
@NoArgsConstructor
@AllArgsConstructor
@DynamoDBTable(tableName = State.TABLE_NAME)
public class State {
   @DynamoDBIgnore
   public static final String TABLE_NAME = "State";
   @DynamoDBHashKey
   private String id;
   private String datasetId;
   private String stateFileName;
   private String generatorAccuracy;
   private String discriminatorAccuracy;
   private String tag;
   private boolean active;
}
```

```
package org.ai.machineart.gan.repository;
import org.ai.machineart.model.DataSet;
import org.socialsignin.spring.data.dynamodb.repository.EnableScan;
import org.springframework.data.repository.CrudRepository;
import java.util.List;
@EnableScan
public interface DataSetRepository extends CrudRepository<DataSet, String> {
   List<DataSet> findByTag(String tag);
}
```

```
package org.ai.machineart.gan.repository;
import org.ai.machineart.gan.model.State;
import org.socialsignin.spring.data.dynamodb.repository.EnableScan;
import org.springframework.data.repository.CrudRepository;
import java.util.List;
@EnableScan
public interface StateRepository extends CrudRepository<State, String> {
   List<State> findByTag(String tag);
}
```

```
package org.ai.machineart.gan.service.impl;
import com.amazonaws.services.s3.AmazonS3;
import lombok.RequiredArgsConstructor;
import org.ai.machineart.exception.DataSetNotFoundException;
import org.ai.machineart.gan.repository.DataSetRepository;
import org.ai.machineart.gan.service.DataSetService;
import org.ai.machineart.model.DataSet;
import org.ai.machineart.utils.ByteArrayConverter;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import java.util.Objects;
@Service
@RequiredArgsConstructor
public class DataSetServiceImpl implements DataSetService {
   @Value("${amazon.s3.bucket.name.dataset}")
   private String bucketName;
   private final DataSetRepository dataSetRepository;
   private final AmazonS3 amazonS3;
   @Override
   public DataSet findById(String id) {
      if (Objects.isNull(id) || id.isBlank()) {
         throw new DataSetNotFoundException(getNotFoundMessageForId(id));
      }
      return dataSetRepository.findById(id)
            .orElseThrow(() -> new DataSetNotFoundException(getNotFoundMessageForId(id)));
   }
```

```
    @Override
    public byte[] findZipFileById(String id) {
        if (!amazonS3.doesObjectExist(bucketName, id + ".zip")) {
            throw new DataSetNotFoundException("Dataset file " + id + ".zip not found");
        }
        return ByteArrayConverter.getByteArrayFromS3Object(amazonS3.getObject(bucketName, id + ".zip"));
    }
    private String getNotFoundMessageForId(String id) {
        if (Objects.isNull(id) || id.isBlank()) {
            return "Please provide dataSetId";
        }
        return "Data set with id " + id + " not found";
    }
}
```

```
package org.ai.machineart.gan.service.impl;
import lombok.RequiredArgsConstructor;
import org.ai.machineart.gan.service.SageMakerService;
import org.springframework.stereotype.Service;
import org.springframework.util.Base64Utils;
import software.amazon.awssdk.services.sagemaker.SageMakerClient;
import software.amazon.awssdk.services.sagemaker.model.*;
import java.nio.charset.StandardCharsets;
import java.util.List;
import java.util.Map;
@Service
@RequiredArgsConstructor
public class SageMakerServiceImpl implements SageMakerService {
    private static final String INSTANCE_NAME = "gan";
    private static final String LIFECYCLE_CONFIG_NAME = "julia";
    private final SageMakerClient sageMakerClient;
    private final Map<ConfigType, String> resourceConfigScripts;
    @Override
    public void startInstance() {
        NotebookInstanceStatus instanceStatus = checkForInstance(INSTANCE_NAME);
        switch (instanceStatus) {
            case STOPPED:
                StartNotebookInstanceRequest startRequest = StartNotebookInstanceRequest.builder()
                        .notebookInstanceName(INSTANCE_NAME)
                        .build();
                sageMakerClient.startNotebookInstance(startRequest);
                break;
            case IN_SERVICE:
            case PENDING:
                throw new IllegalStateException("Instance has already been started");
            case STOPPING:
                throw new IllegalStateException("Instance is stopping, starting is prohibited");
            case FAILED:
            case DELETING:
            case UPDATING:
            case UNKNOWN_TO_SDK_VERSION:
                throw new IllegalStateException("Wrong state of instance: " + instanceStatus);
        }
    }
    @Override
    public void stopInstance() {
        NotebookInstanceStatus instanceStatus = checkForInstance(INSTANCE_NAME);
        switch (instanceStatus) {
            case IN_SERVICE:
                StopNotebookInstanceRequest stopRequest = StopNotebookInstanceRequest.builder()
                        .notebookInstanceName(INSTANCE_NAME)
                        .build();
                sageMakerClient.stopNotebookInstance(stopRequest);
```

```
            break;
        case PENDING:
            throw new IllegalStateException("Instance is starting, stopping is prohibited");
        case FAILED:
        case DELETING:
        case UPDATING:
        case UNKNOWN_TO_SDK_VERSION:
            throw new IllegalStateException("Wrong state of instance: " + instanceStatus);
        default:
            break;
      }
    }
    @Override
    public void loadStartConfig(ConfigType type) {
      checkForLifecycleConfig(LIFECYCLE_CONFIG_NAME);
      UpdateNotebookInstanceLifecycleConfigRequest request = UpdateNotebookInstanceLifecycleConfigRequest
          .builder()
          .notebookInstanceLifecycleConfigName(LIFECYCLE_CONFIG_NAME)
          .onStart(NotebookInstanceLifecycleHook.builder()
              .content(Base64Utils.encodeToString(resourceConfigScripts
                  .get(type).getBytes(StandardCharsets.UTF_8)))
              .build())
          .onCreate(NotebookInstanceLifecycleHook.builder()
              .content(Base64Utils.encodeToString(resourceConfigScripts
                  .get(ConfigType.CREATION).getBytes(StandardCharsets.UTF_8)))
              .build())
          .build();
      sageMakerClient.updateNotebookInstanceLifecycleConfig(request);
    }
    private NotebookInstanceStatus checkForInstance(String instanceName) {
      List<NotebookInstanceStatus> ganInstances =
sageMakerClient.listNotebookInstances().notebookInstances().stream()
          .filter(el -> el.notebookInstanceName().equals(instanceName))
          .map(NotebookInstanceSummary::notebookInstanceStatus)
          .toList();
      if (ganInstances.size() != 1) {
        throw new RuntimeException("There is no SageMaker instance for testing");
      }
      return ganInstances.get(0);
    }
    private void checkForLifecycleConfig(String lifecycleConfigName) {
      long lifecycleConfigsNum =
sageMakerClient.listNotebookInstanceLifecycleConfigs().notebookInstanceLifecycleConfigs().stream()
          .filter(el -> el.notebookInstanceLifecycleConfigName().equals(lifecycleConfigName))
          .count();
      if (lifecycleConfigsNum != 1) {
        throw new RuntimeException("There is no SageMaker lifecycle config with name " + lifecycleConfigName);
      }
    }
}

package org.ai.machineart.gan.service.impl;
import org.ai.machineart.gan.service.ZipService;
import org.springframework.stereotype.Service;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.util.Map;
import java.util.zip.ZipEntry;
import java.util.zip.ZipOutputStream;
@Service
public class ZipServiceImpl implements ZipService {
    @Override
```

```
    public ByteArrayInputStream zipToSingleByteArray(Map<String, byte[]> fileNameWithContent) {
        try {
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            ZipOutputStream zos = new ZipOutputStream(baos);
            fileNameWithContent.entrySet().forEach(el -> putPictureToZip(zos, el));
            zos.close();
            return new ByteArrayInputStream(baos.toByteArray());
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
    private void putPictureToZip(ZipOutputStream zos, Map.Entry<String, byte[]> fileNameWithContentEntry) {
        try {
            ZipEntry entry = new ZipEntry(fileNameWithContentEntry.getKey());
            entry.setSize(fileNameWithContentEntry.getValue().length);
            zos.putNextEntry(entry);
            zos.write(fileNameWithContentEntry.getValue());
            zos.closeEntry();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
}

package org.ai.machineart.gan.service.impl;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import lombok.RequiredArgsConstructor;
import org.ai.machineart.exception.StateNotFoundException;
import org.ai.machineart.gan.model.State;
import org.ai.machineart.gan.repository.StateRepository;
import org.ai.machineart.gan.service.StateService;
import org.ai.machineart.utils.ByteArrayConverter;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import java.io.ByteArrayInputStream;
import java.io.InputStream;
import java.util.List;
import java.util.Map;
import java.util.Objects;
import java.util.stream.Collectors;
import java.util.stream.StreamSupport;
@Service
@RequiredArgsConstructor
public class StateServiceImpl implements StateService {
    private static final String ACTIVE_STATE_FILE_NAME = "dcgan.bson";
    private static final String PROPERTIES_FILE_NAME = "dcgan.json";
    private static final String ACTIVE_TRAIN_FILE_NAME = "dcgan.zip";
    private static final String DEACTIVATION_FILE_NAME = "dcgan.jpg";
    @Value("${amazon.s3.bucket.name.state}")
    private String bucketName;
    private final StateRepository stateRepository;
    private final AmazonS3 amazonS3;
    @Override
    public State save(State state) {
        return stateRepository.save(state);
    }
    @Override
    public String saveToS3(State state, byte[] stateFile) {
```

```
        amazonS3.putObject(bucketName, state.getStateFileName(), new ByteArrayInputStream(stateFile), new
ObjectMetadata());
        amazonS3.putObject(bucketName, state.getId() + "/", InputStream.nullInputStream(), new ObjectMetadata());
        return state.getStateFileName();
    }
    @Override
    public void activateTesting(State state) {
        amazonS3.copyObject(bucketName, state.getStateFileName(), bucketName, ACTIVE_STATE_FILE_NAME);
        state.setActive(true);
        update(state);
    }
    @Override
    public void activateTraining(State state, byte[] datasetFile, int epochs) {
        if (amazonS3.doesObjectExist(bucketName, state.getStateFileName())) {
            amazonS3.copyObject(bucketName, state.getStateFileName(), bucketName, ACTIVE_STATE_FILE_NAME);
        }
        amazonS3.putObject(bucketName, ACTIVE_TRAIN_FILE_NAME, new ByteArrayInputStream(datasetFile),
new ObjectMetadata());
        amazonS3.putObject(bucketName, state.getId() + "/", InputStream.nullInputStream(), new ObjectMetadata());
        try {
            ObjectMapper mapper = new ObjectMapper();
            JsonNode node = mapper.createObjectNode().put("epoch", epochs);
            String propertiesJson = mapper.writeValueAsString(node);
            amazonS3.putObject(bucketName, PROPERTIES_FILE_NAME, propertiesJson);
        } catch (JsonProcessingException ignored) {
        }
        state.setActive(true);
        update(state);
    }
    @Override
    public boolean deactivate(State state) {
        try {
            long deactivationFilesNum = amazonS3.listObjects(bucketName).getObjectSummaries().stream()
                    .filter(el -> el.getKey().equalsIgnoreCase(DEACTIVATION_FILE_NAME))
                    .count();
            if (deactivationFilesNum != 1) { return false; }
            String prefix = state.getId() + "/";
            long fileNum = amazonS3.listObjects(bucketName, prefix).getObjectSummaries().stream()
                    .filter(el -> el.getKey().length() > prefix.length())
                    .count() + 1;
            String imgKey = state.getId() + "/" + fileNum + ".jpg";
            amazonS3.copyObject(bucketName, ACTIVE_STATE_FILE_NAME, bucketName, state.getStateFileName());
            amazonS3.deleteObject(bucketName, ACTIVE_STATE_FILE_NAME);
            amazonS3.copyObject(bucketName, DEACTIVATION_FILE_NAME, bucketName, imgKey);
            amazonS3.deleteObject(bucketName, DEACTIVATION_FILE_NAME);
            amazonS3.deleteObject(bucketName, ACTIVE_TRAIN_FILE_NAME);
            amazonS3.deleteObject(bucketName, PROPERTIES_FILE_NAME);
            state.setActive(false);
            update(state);
            return true;
        } catch (Exception e) {
            return false;
        }
    }
    @Override
    public State update(State state) {
        String id = state.getId();
        if (Objects.isNull(id) || id.isBlank() || stateRepository.findById(id).isEmpty()) {
            throw new StateNotFoundException(getNotFoundMessageForId(id));
        }
        return stateRepository.save(state);
    }
    @Override
```

```
    public State findById(String id) {
        if (Objects.isNull(id) || id.isBlank()) {
            throw new StateNotFoundException(getNotFoundMessageForId(id));
        }
        return stateRepository.findById(id).orElseThrow(() ->
          new StateNotFoundException(getNotFoundMessageForId(id)));
    }
    @Override
    public Map<String, byte[]> findGeneratedPicturesById(String id) {
        stateRepository.findById(id).orElseThrow(() -> new StateNotFoundException(getNotFoundMessageForId(id)));
        final String prefix = id + "/";
        return amazonS3.listObjects(bucketName, prefix).getObjectSummaries().stream()
                .filter(el -> el.getKey().length() > prefix.length())
                .map(el -> amazonS3.getObject(bucketName, el.getKey()))
                .map(el -> Map.entry(el.getKey().replace(prefix, ""), ByteArrayConverter.getByteArrayFromS3Object(el)))
                .collect(Collectors.toMap(Map.Entry::getKey, Map.Entry::getValue, (el1, el2) -> el2));
    }
    @Override
    public List<State> findByTag(String tag) {
        if (Objects.isNull(tag) || tag.isBlank()) {
            throw new StateNotFoundException(getNotFoundMessageForTag(tag));
        }
        List<State> states = stateRepository.findByTag(tag);
        if (states.isEmpty()) {
            throw new StateNotFoundException(getNotFoundMessageForTag(tag));
        }
        return states;
    }
    @Override
    public List<State> findAll() {
        return StreamSupport.stream(stateRepository.findAll().spliterator(), false).collect(Collectors.toList());
    }
    @Override
    public void deleteById(String id) {
        Optional<State> state = stateRepository.findById(id);
        if (state.isEmpty()) {
            throw new StateNotFoundException(getNotFoundMessageForId(id));
        }
        stateRepository.deleteById(id);
        amazonS3.deleteObject(bucketName, state.get().getStateFileName());
        String[] picturesForDelete = amazonS3.listObjects(bucketName, state.get().getId() + "/")
                .getObjectSummaries().stream()
                .map(S3ObjectSummary::getKey)
                .toList()
                .toArray(new String[]{});
        DeleteObjectsRequest deleteDirectoryRequest =
             new DeleteObjectsRequest(bucketName).withKeys(picturesForDelete);
        amazonS3.deleteObjects(deleteDirectoryRequest);
}
    private String getNotFoundMessageForId(String id) {
        if (Objects.isNull(id) || id.isBlank()) {
            return "Please provide stateId";
        }
        return "State with id " + id + " not found";
    }
    private String getNotFoundMessageForTag(String tag) {
        if (Objects.isNull(tag) || tag.isBlank()) {
            return "Please provide tag of arts";
        }
        return "States of GAN for tag " + tag + " not found";
    }
}
```

ДОДАТОК Б

ПРИКЛАДИ ЗГЕНЕРОВАНИХ ЗОБРАЖЕНЬ

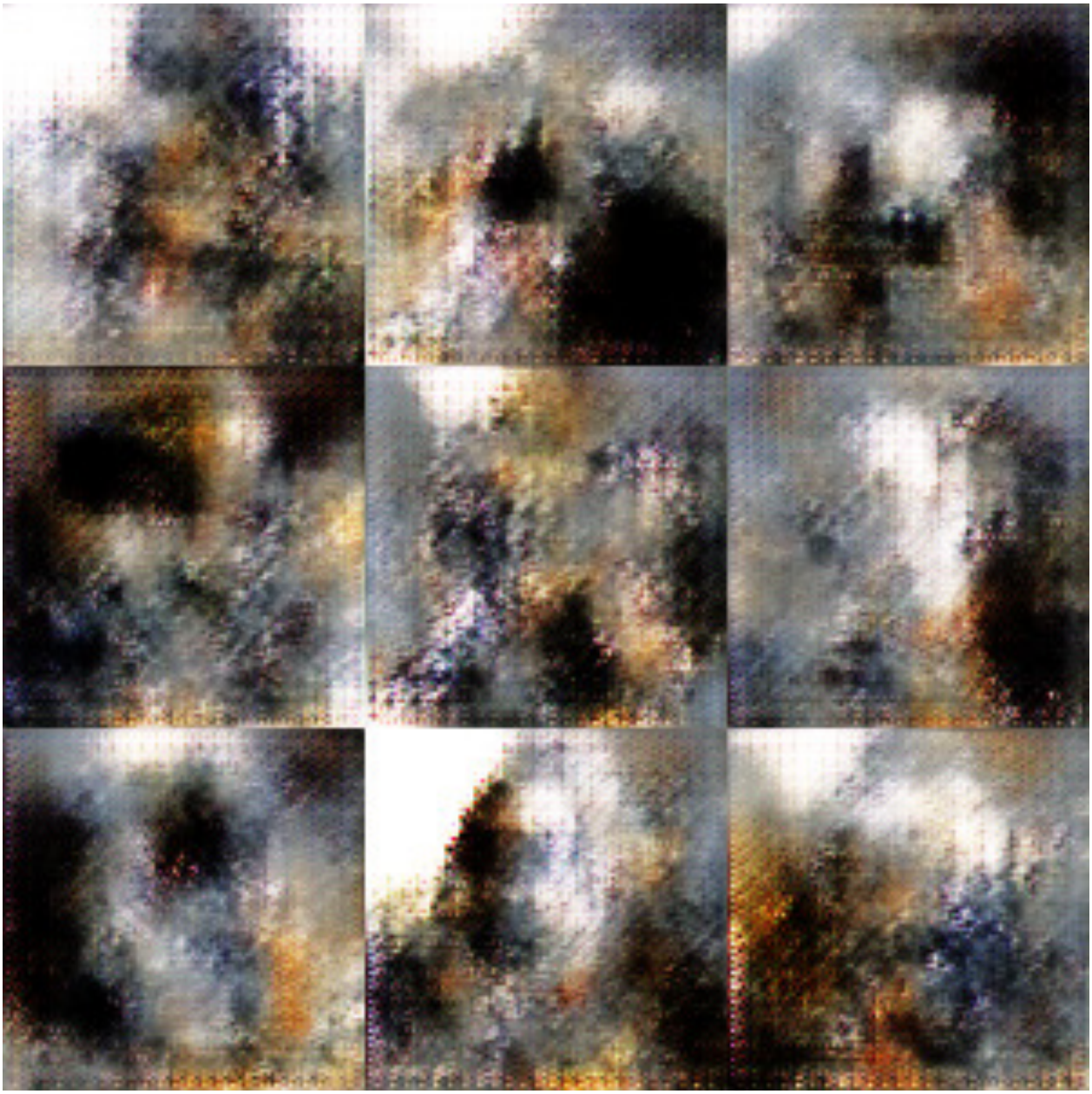

Рис. Б.1. Приклад на 1000 епохах

Рис. Б.2. Приклад на 10000 епохах

Рис. Б.3. Приклад на 20000 епохах

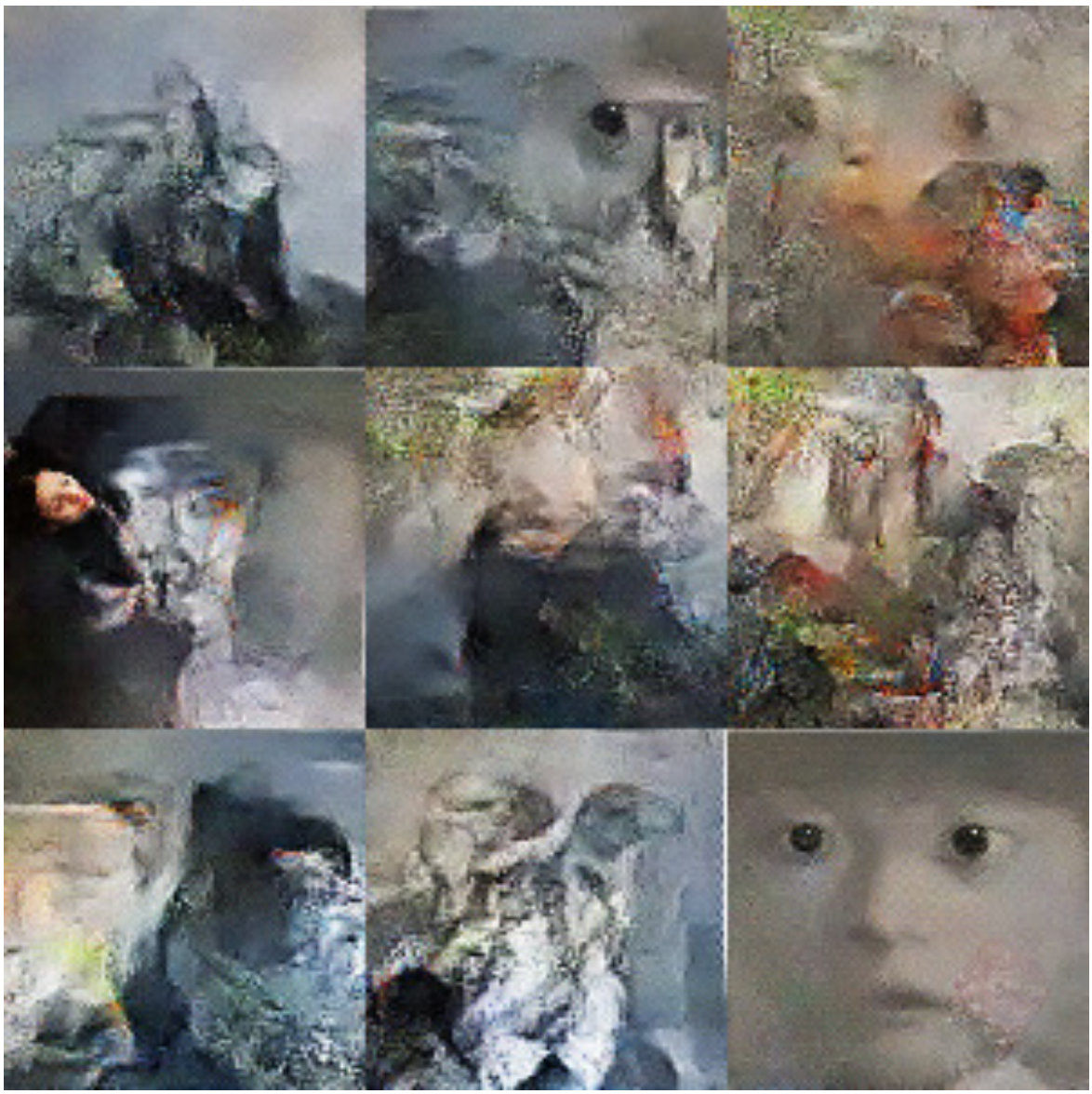

Рис. Б.3. Приклад на 30000 епохах

ДОДАТОК В

ВІДГУК
керівника економічного розділу
на кваліфікаційну роботу бакалавра
на тему:
«Методи і моделі для реалізації концепції машинного мистецтва»
студента групи 122-19ск-1 Руксова Євгенія Вікторовича

Керівник економічного розділу
доцент каф. ПЕП та ПУ, к.е.н. **Л. В. Касьяненко**

ДОДАТОК Г

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
ПЗ_Руксов.doc	Пояснювальна записка до кваліфікаційної роботи, документ MS Word
ПЗ_Руксов.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Program.zip	Архів. Містить коди програми і откомпільовану програму
Презентація	
Презентація_Руксов.ppt	Презентація кваліфікаційної роботи