НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри комп'ютерних наук**
_____ Ігор БОЛБОТ	**_____ Белла ГОЛУБ**
(підпис)	(підпис)
«___» _____ 2026 р.	«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Програмне забезпечення редактора двовимірної графіки та анімації»
Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми к.т.н., доцент	_____ (підпис)	**Ганна ВАЙГАНГ**
Керівник бакалаврської кваліфікаційної роботи	_____ (підпис)	**Максим НЕДЬОШЕВ**
Консультант бакалаврської кваліфікаційної роботи д.е.н., професор	_____ (підпис)	**Роман РУДЕНСЬКИЙ**
Виконав	_____ (підпис)	**Вадим ГЕРАСИМЕНКО**

КИЇВ – 2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук
к.т.н., доцент _____ Белла ГОЛУБ
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧУ

Герасименко Вадиму Миколайовичу
(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»
Тема бакалаврської кваліфікаційної роботи
«Програмне забезпечення редактора двовимірної графіки та анімації»
затверджена наказом від «19» грудня 2025 р. № 3196 «С»

Термін подання завершеної роботи на кафедру «22» травня 2026 р.

Вихідні дані до бакалаврської кваліфікаційної роботи:
Матеріали аналізу існуючих графічних редакторів, засоби веб-розробки Django, JavaScript, HTML5 Canvas, PostgreSQL, Redis, Docker.
Перелік питань, що підлягають дослідженню:
Аналіз предметної області та постановка завдання, проектування, інформаційного забезпечення, розробка програмного забезпечення редактора двовимірної графіки та анімації, в провадження та експлуатація програмного забезпечення
Перелік графічних документів (за необхідності).

Дата видачі завдання « 19 » грудня 2025 р.

Керівник бакалаврської кваліфікаційної роботи _____ **Максим НЕДЬОШЕВ**
(підпис)

Консультант бакалаврської кваліфікаційної роботи _____ **Роман РУДЕНСЬКИЙ**
д.е.н., професор (підпис)

Завдання взяв до виконання _____ **Вадим ГЕРАСИМЕНКО**
(підпис)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – База даних

ПЗ – Програмне забезпечення

СУБД – Система управління базами даних

СКБД – Система керування базами даних

2D – Двовимірна графіка

1NF (First Normal Form) – Перша нормальна форма

2NF (Second Normal Form) – Друга нормальна форма

3NF (Third Normal Form) – Третя нормальна форма

ACID (Atomicity, Consistency, Isolation, Durability) – Властивості транзакцій бази даних

API (Application Programming Interface) – Інтерфейс прикладного програмування

ASGI (Asynchronous Server Gateway Interface) – Асинхронний серверний інтерфейс для Python-застосунків

AVI (Audio Video Interleave) – Формат відеофайлів

BMP (Bitmap) – Формат растрових зображень

CSS (Cascading Style Sheets) – Каскадні таблиці стилів

CSRF (Cross-Site Request Forgery) – Міжсайтова підробка запиту

CSV (Comma-Separated Values) – Формат табличних даних

EPS (Encapsulated PostScript) – Формат векторної графіки

ER-діаграма (Entity-Relationship diagram) – Діаграма «сутність-зв’язок»

ES6 (ECMAScript 2015) – Стандарт мови JavaScript

FFmpeg – Інструмент для обробки мультимедійних даних

FK (Foreign Key) – Зовнішній ключ

FPS (Frames Per Second) – Кількість кадрів за секунду

GIF (Graphics Interchange Format) – Графічний формат із підтримкою анімації

GIMP (GNU Image Manipulation Program) – Графічний редактор

GPL (General Public License) – Загальна публічна ліцензія

HTML (HyperText Markup Language) – Мова гіпертекстової розмітки

HTML5 – П’ята версія стандарту HTML

HTTP (HyperText Transfer Protocol) – Протокол передавання гіпертексту

HTTPS (HyperText Transfer Protocol Secure) – Захищений протокол передавання гіпертексту

HSTS (HTTP Strict Transport Security) – Механізм примусового використання HTTPS

JPEG (Joint Photographic Experts Group) – Формат стиснення растрових зображень

JSON (JavaScript Object Notation) – Текстовий формат обміну даними

JSONB (JSON Binary) – Бінарний формат зберігання JSON-даних

JWT (JSON Web Token) – Токен для безпечної передачі даних

MP4 (MPEG-4 Part 14) – Формат відеофайлів

MVT (Model-View-Template) – Архітектурний шаблон Django

NoSQL (Not Only SQL) – Підхід до нереляційного зберігання даних

NTSC (National Television System Committee) – Телевізійний стандарт

OAuth (Open Authorization) – Відкритий стандарт авторизації

ORM (Object-Relational Mapping) – Об’єктно-реляційне відображення

PAL (Phase Alternating Line) – Телевізійний стандарт

PK (Primary Key) – Первинний ключ

PNG (Portable Network Graphics) – Растровий графічний формат

PSD (Photoshop Document) – Формат файлів Adobe Photoshop

REST (Representational State Transfer) – Архітектурний стиль взаємодії веб-сервісів

REST API – API, побудований за принципами REST

SQL (Structured Query Language) – Структурована мова запитів

SSL (Secure Sockets Layer) – Протокол захищеного з’єднання

SVG (Scalable Vector Graphics) – Масштабована векторна графіка

TTL (Time To Live) – Час життя запису або блокування

UI (User Interface) – Інтерфейс користувача

UML (Unified Modeling Language) – Уніфікована мова моделювання

URL (Uniform Resource Locator) – Уніфікований покажчик ресурсу

VP9 – Відеокодек

WebM – Відкритий відеоформат для вебсередовища

WebSocket – Протокол двостороннього обміну даними в реальному часі

XCF – Формат файлів графічного редактора GIMP

XML (Extensible Markup Language) – Розширювана мова розмітки

XSS (Cross-Site Scripting) – Міжсайтове виконання сценаріїв

ВСТУП

Сучасний розвиток цифрових технологій у сферах кінематографу, комп’ютерних ігор, веб-дизайну, реклами та освіти неможливий без використання двовимірної графіки та анімації. Від традиційних мальованих мультфільмів до інтерактивних веб-інтерфейсів – скрізь потрібні інструменти, які дозволяють швидко створювати, редагувати та відтворювати зображення, що змінюються в часі. Водночас, попри велику кількість існуючих графічних редакторів (Adobe Photoshop, Krita, GIMP, Aseprite, Pencil2D тощо), багато з них є або надто складними для початківців, або не підтримують колективної роботи, або вимагають встановлення на локальний комп’ютер та не мають зручних веб-інтерфейсів для спільної творчості.

Особливо гостро проблема спільного редагування постає у навчальних закладах, невеликих анімаційних студіях та розподілених командах, де важливо мати можливість одночасно працювати над одним проєктом, бачити дії один одного в реальному часі та швидко обмінюватися коментарями. Саме тому створення веб-орієнтованого програмного забезпечення редактора двовимірної графіки та анімації з підтримкою колаборативних функцій є актуальним науково-практичним завданням, що має значний попит як у професійному середовищі, так і в освітній сфері. Крім того, сучасний рівень розвитку веб-технологій (HTML5 Canvas, WebSockets, Django Channels, Redis) дозволяє створювати повноцінні графічні редактори безпосередньо в браузері, які за функціоналом не поступаються десктопним аналогам, але при цьому є доступними з будь-якого пристрою без встановлення додаткового програмного забезпечення. Таким чином, розробка власного програмного забезпечення редактора двовимірної графіки та анімації є не лише актуальною, але й технічно реалізованою на сучасному рівні.

Метою даної роботи є розробка програмного забезпечення для створення та редагування двовимірної графіки та покадрової анімації, яке забезпечує можливість спільної роботи кількох користувачів у реальному часі, зберігання проєктів на сервері, експорт результатів у поширені формати та інтуїтивно зрозумілий інтерфейс.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Проаналізувати предметну область.
2. Спроєктувати архітектуру веб-застосунку.
3. Розробити логічну модель даних.
4. Реалізувати основні функції графічного редактора.
5. Впровадити колаборативну складову.
6. Реалізувати функцію експорту анімації.
7. Провести тестування розробленого програмного забезпечення.

Об’єктом дослідження є процес створення та редагування двовимірної графіки й анімації засобами сучасних веб-технологій. Предметом дослідження є методи та програмні засоби реалізації графічного редактора з підтримкою колаборативного режиму, збереженням стану проєкту, версіонуванням шарів та експортом результатів. Методи та технології дослідження. У роботі використано такі методи: аналіз предметної області та порівняльний аналіз аналогів, об’єктно-орієнтоване проєктування, моделювання даних, а також метод експериментального тестування. Структура та обсяг роботи.

Пояснювальна записка складається зі вступу, чотирьох розділів основної частини, висновків, списку використаних джерел. У першому розділі проведено аналіз предметної області: розглянуто основні поняття двовимірної графіки та анімації, принципи роботи графічних редакторів, виконано порівняльний аналіз існуючих рішень, сформульовано функціональні та нефункціональні вимоги до програмного забезпечення, що розробляється. Другий розділ присвячено проєктуванню інформаційного забезпечення: розроблено ER-діаграму бази даних, обґрунтовано вибір системи управління інформаційним забезпеченням, описано структуру інформаційної бази, а також схему фізичного зберігання

даних. Третій розділ містить опис розробки програмного забезпечення: визначено базові сутності предметної області, змодельовано структуру та взаємодію об’єктів, обґрунтовано вибір стеку розробки, наведено блок-схеми ключових алгоритмів та фрагменти коду основних програмних модулів. Також описано реалізацію інтерфейсу користувача. Четвертий розділ висвітлює питання впровадження та експлуатації системи: описано процес тестування, наведено вимоги до апаратного та програмного забезпечення, а також склад інсталяційного пакету. У висновках підведено підсумки роботи, оцінено досягнення поставленої мети, окреслено перспективи подальшого розвитку системи.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз редакторів двовимірної графіки та анімації як предметної області

Предметна область створення та редагування двовимірної графіки та анімації охоплює широке коло програмних продуктів, теоретичних концепцій, методів і технологій, які використовуються для створення, обробки та відтворення плоских зображень та їх послідовностей. Дана сфера має значне практичне застосування в кінематографі, комп'ютерних іграх, веб-дизайні, рекламі, поліграфії, освіті та багатьох інших галузях. Розуміння сутності та структури цієї предметної області є необхідною передумовою для розробки спеціалізованого програмного забезпечення. 2D-графіка – це область комп'ютерної графіки, яка оперує зображеннями, що мають лише два виміри: ширину та висоту. За способом подання та зберігання зображень двомірна графіка традиційно поділяється на два основних типи: растрову та векторну.

1. Растрова графіка представляє зображення у вигляді матриці пікселів, кожен з яких має власні координати та колір. Основними характеристиками растрового зображення є його роздільна здатність та глибина кольору. Растровий підхід є природним для більшості графічних редакторів, оскільки він безпосередньо відповідає способу виведення зображення на екран монітора або друку на принтері. До переваг растрової графіки належать можливість відтворення складних кольорових переходів, реалістичних текстур та фотографічної точності. Основним недоліком є залежність якості зображення від роздільної здатності – при масштабуванні растрове зображення втрачає чіткість, а також великий обсяг файлів для зображень високої роздільної здатності [1].

2. Векторна графіка описує зображення за допомогою геометричних примітивів (точок, ліній, кривих, багатокутників) та математичних формул, що їх визначають. Кожен об’єкт має свої атрибути – колір заливки, колір контуру, товщину лінії тощо. На відміну від растрової, векторна графіка дозволяє необмежено масштабувати зображення без втрати якості, оскільки при зміні розміру примітиви перераховуються заново. Векторні зображення зазвичай мають значно менший обсяг порівняно з растровими аналогами. Основними недоліками є складність або неможливість відтворення фотореалістичних зображень та більш висока обчислювальна складність при рендерингу.

Анімація – це техніка створення ілюзії руху шляхом швидкої послідовної зміни нерухомих зображень, які називаються кадрами. Людське око сприймає окремі кадри як безперервний рух за умови, що частота їх зміни перевищує приблизно 10-12 кадрів на секунду. Стандартними значеннями частоти кадрів для різних медіа є 24 кадри на секунду (кінематограф), 25 (PAL телебачення), 30 (NTSC телебачення) та вищі (60-120 кадрів на секунду для відеоігор та дисплеїв високої частоти оновлення). У двомірній комп’ютерній анімації виділяють кілька основних технік та підходів.

1. Покадрова анімація є найбільш класичним та трудомістким методом, при якому кожен окремий кадр створюється художником окремо. Цей підхід забезпечує найбільшу гнучкість та виразність, дозволяє створювати органічні, плавні рухи з довільними деформаціями об'єктів. Однак він потребує значних часових та трудових ресурсів. Покадрова анімація є основою для традиційної мальованої анімації, а також широко використовується в комп’ютерних іграх у жанрі піксельної графіки [2].

2. Анімація за ключовими кадрами базується на заданні ключових положень об'єктів у певні моменти часу, після чого програмне забезпечення автоматично розраховує та генерує проміжні кадри. Цей метод значно зменшує обсяг ручної роботи та є стандартом для більшості сучасних анімаційних систем, особливо при роботі з векторною графікою. Автоматична генерація проміжних

кадрів може бути лінійною, сплайновою або використовувати інші функції інтерполяції.

3. Анімація на основі скелета використовується переважно для анімації персонажів. При цьому створюється ієрархічна структура кісток (скелет), яка прив’язується до частин зображення (шкіри). Аніматор задає рухи кісток, а програмне забезпечення автоматично деформує прив’язане зображення відповідно до трансформацій скелета. Цей підхід є надзвичайно ефективним для створення складних рухів персонажів з мінімальними витратами ресурсів.

У контексті створення програмного забезпечення редактора двовимірної графіки та анімації ключовими поняттями також є «шар», «таймлайн» та «інструмент малювання». Шар – це окремий рівень, на якому розміщуються графічні об'єкти. Шари можуть накладатися один на одного, утворюючи композицію. Кожен шар зазвичай має власні параметри прозорості, видимості, режиму накладання. Використання шарів дозволяє редагувати елементи зображення незалежно один від одного, що значно спрощує роботу зі складними сценами. Приклад розділення анімації на декілька шарів наведено на рис. 1.1.

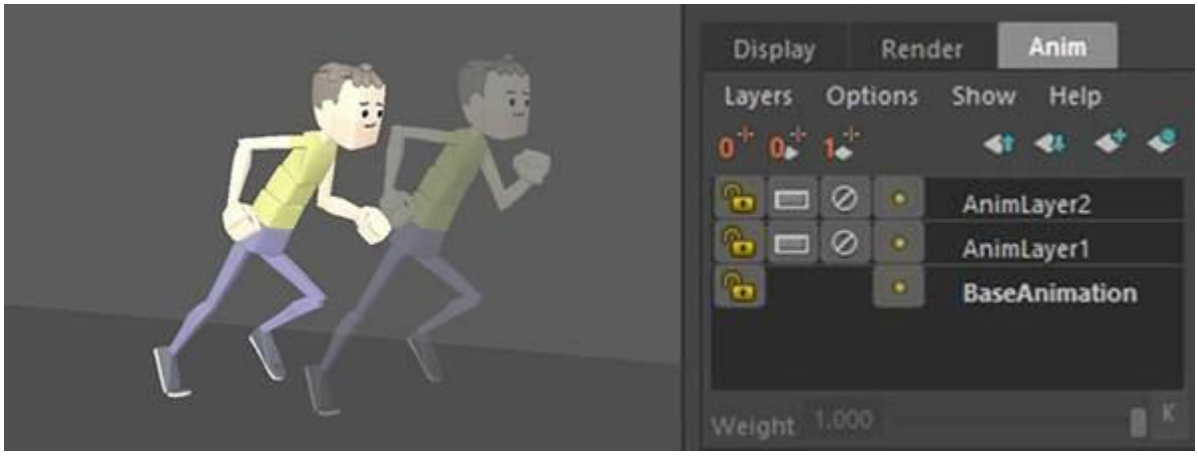


Рис. 1.1 – Розділення анімації на декілька шарів

Таймлайн – це візуальний інтерфейс для роботи з анімацією, який відображає послідовність кадрів (або ключових кадрів) у часі. Таймлайн дозволяє переміщатися між кадрами, додавати, видаляти або дублювати кадри, змінювати їх порядок, встановлювати тривалість та налаштовувати параметри відтворення. Приклад використання таймлайну в анімації наведено на рис. 1.2.

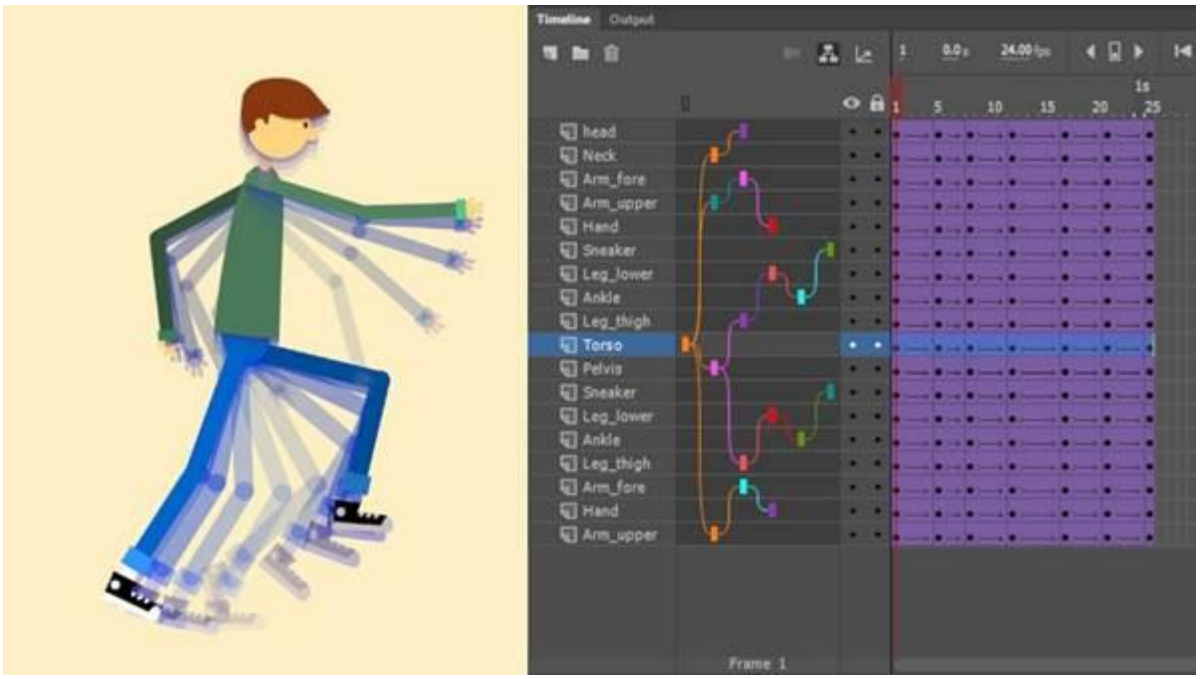


Рис. 1.2 – Приклад використання таймлайну в анімації бігу

Інструменти малювання – набір засобів для безпосереднього створення графічного вмісту. До типових інструментів належать пензлик з можливістю зміни розміру, форми та твердості, ластик, заливка, інструменти створення геометричних фігур, піпетка для вибору кольору з зображення, виділення для роботи з фрагментами зображення. Приклад типової панелі інструментів малювання подано на рис. 1.3.

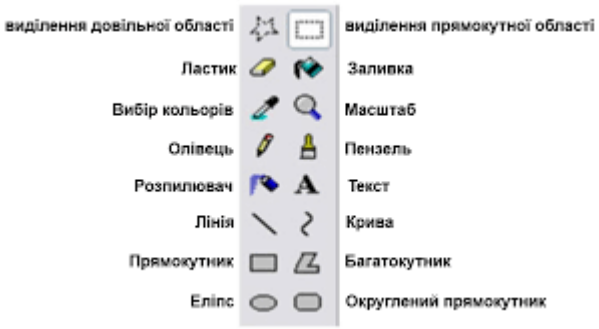


Рис. 1.3 – Приклад типової панелі інструментів малювання

Важливою характеристикою графічних редакторів є підтримка форматів збереження. Нативний формат редактора зазвичай зберігає всю інформацію про шари, анімаційні доріжки, налаштування інструментів тощо. Формати експорту призначені для кінцевого використання зображень та анімації – це можуть бути

растрові формати (PNG, JPEG, BMP, GIF), векторні (SVG, EPS) або відеоформати (MP4, WebM, AVI) [3].

Отже, предметна область редакторів двовимірної графіки та анімації є багатогранною та включає в себе теоретичні основи подання зображень, техніки створення анімації, а також прикладні аспекти реалізації інструментів редагування, роботи з шарами, таймлайном та форматами даних. Подальший аналіз цієї предметної області дозволить перейти до моделювання ключових процесів та порівняльного аналізу існуючих програмних рішень.

1.2 Моделювання предметної області

Моделювання предметної області є важливим етапом аналізу, оскільки дозволяє формалізувати знання про систему, визначити її межі, учасників та основні процеси. У даній роботі для моделювання обрано мову UML, яка є стандартом для візуалізації, специфікації та документування програмних систем. У цьому підрозділі розглянуто діаграму прецедентів та діаграму діяльності.

Діаграма прецедентів призначена для опису функціональних вимог до системи з точки зору зовнішніх користувачів (акторів). Вона показує, які функції (варіанти використання) система надає кожному актору, та дозволяє визначити межі системи. Детальний аналіз розробленої діаграми варіантів використання дозволяє чітко розмежувати зони відповідальності та функціональні можливості користувачів у межах архітектури веб-програми. Кожен актор уособлює певний набір прав доступу, який валідується серверною частиною системи та динамічно обробляється клієнтським сценарієм.

Детальна характеристика системних акторів:

- Гість (Неавторизований користувач): Даний актор представляє точку входу в систему. Його взаємодія з веб-ресурсом обмежена базовим HTTP-протоколом. Головне цільове завдання цього актора –

проходження процедури ідентифікації для отримання статусу авторизованого користувача.

- Власник проєкту: Ключовий актор системи, який володіє повним набором прав щодо створеного ним графічного та анімаційного контенту. Він виступає модератором сесії та ініціатором розподіленого робочого простору. Його дії безпосередньо впливають на стан бази даних та активні WebSocket-групи.
- Співавтор: Авторизований користувач, який володіє правами оперативного редагування графіки в межах конкретної сесії, куди його було долучено. Він має доступ до інструментарію малювання та обміну реалтайм-повідомленнями, проте позбавлений адміністративних функцій (наприклад, він не може остаточно видалити проєкт із бази даних або керувати правами інших учасників).

Варіанти використання:

1. Реєстрація та автентифікація

Даний прецедент є базовою точкою входу для систем, що підтримують збереження користувацьких даних або мережеву взаємодію. Забезпечує процес створення унікального цифрового профілю користувача в системі (введення ідентифікаційних даних) та подальшу перевірку автентичності (вхід за паролем або токеном). Сценарій використання: Користувач ініціює сесію, система валідує права доступу і надає доступ до персоналізованого простору, сховища проєктів та мережевих функцій [4].

2. Створення проєкту

Варіант використання, який відповідає за ініціалізацію нового робочого простору та встановлення первинних глобальних метаданих. Дозволяє користувачеві визначити базові параметри майбутньої анімації: геометричні розміри полотна (роздільну здатність у пікселях), базову частоту кадрів (FPS) та колірний простір. Сценарій використання: Система резервує обчислювальні ресурси під нову сутність, створює початковий порожній кадр із нульовим шаром і відкриває візуальний інтерфейс редактора.

3. Перегляд списку власних проєктів

Прецедент, спрямований на забезпечення менеджменту та навігації по створених раніше файлах чи хмарних сховищах. Надає інтерфейс каталогу (панелі керування), де відображаються всі доступні користувачеві роботи (як особисті, так і ті, до яких йому надано спільний доступ). Сценарій використання: Користувач аналізує метадані (дату модифікації, прев'ю кадру), здійснює пошук, фільтрацію або видалення застарілих проєктів.

4. Редагування графічного вмісту

Центральний і найбільш складний прецедент предметної області, який акумулює в собі всі інструменти безпосередньої взаємодії з цифровим полотном. За принципом декомпозиції він включає в себе такі обов'язкові сценарії:

- Малювання на полотні: Сценарій послідовної зміни матриці пікселів за допомогою інтерактивних інструментів. Користувач вибирає інструмент (пензель, гумка, фігури, заливка) та параметри (колір, товщина), а система динамічно прораховує колірні зміни на екрані. Сюди ж відноситься сценарій виділення областей (прямокутник, ласо, чарівна паличка) для обмеження зони редагування.
- Керування багатошаровою структурою: Маніпуляції зі стопкою прозорих площин. Сценарій передбачає створення нових шарів, зміну їхнього відносного розташування, налаштування прозорості та перемикання видимості для відокремлення статичних елементів від динамічних.

5. Спільна робота у реальному часі

Спеціалізований прецедент, що описує логіку багатокористувацької взаємодії в межах одного проєкту. Він складається з таких взаємопов'язаних процесів:

- Трансляція дій та курсору: Сценарій безперервної передачі координат покажчика та команд малювання від одного користувача до інших. Кожен учасник сесії бачить графічні маркери (курсори) своїх колег та результати їхнього малювання без затримок.

- Коментування кадрів: Можливість залишати контекстні текстові нотатки або технічні завдання з прив'язкою до конкретного моменту часу (кадру) на таймлайні для швидкої комунікації всередині команди.

6. Керування учасниками

Прецедент, що відповідає за розмежування прав доступу та адміністрування спільного робочого простору. Надає інструменти для делегування прав користувачам. Сценарій передбачає генерацію унікальних посилань-запрошень, додавання нових співавторів за їхніми ідентифікаторами або примусове відключення учасників від сесії з відкликом прав на редагування.

7. Експорт анімації

Фінальний прецедент життєвого циклу створення проєкту, що відповідає за конвертацію внутрішнього представлення даних у сумісні формати. Запуск процесу фінального рендерингу (композитингу). Сценарій використання: Система покроково зчитує всі кадри, накладає шари відповідно до їхніх коефіцієнтів прозорості та режимів змішування кольорів, формує послідовність статичних зображень і кодує їх у кінцевий файл: растровий медіа-потік (відео у форматах MP4/WebM), анімований графічний файл (GIF) або окремий пакет статичних кадрів (PNG-секвенція) з урахуванням заданої частоти FPS [5].

Детальніше, діаграма прецедентів показана в додатку Б.

Діаграма діяльності UML використовується для моделювання динамічних аспектів системи – потоків управління та даних між діями. Вона показує послідовність виконання операцій, розгалуження, паралельне виконання та синхронізацію потоків. У даній роботі діаграму діяльності побудовано для процесу «Редагування анімації в реальному часі з колаборативною підтримкою», оскільки це є ядром функціональності системи. Процес редагування анімації починається з того, що користувач (який має права редактора або власника проєкту) обирає інструмент малювання (пензлик, гумка, заливка, фігура) та починає малювати на активному шарі поточного кадру. Система перевіряє, чи має користувач блокування на цей шар (оскільки два редактори не можуть одночасно редагувати один шар). Якщо блокування відсутнє та користувач не є

його власником – система намагається отримати блокування через WebSocket. Якщо отримати блокування не вдалося (шар заблоковано іншим), система показує повідомлення про неможливість редагування. Якщо блокування успішне або вже належить користувачеві, система дозволяє малювання. Під час малювання система виконує дві паралельні дії: локальне відображення штриха на полотні користувача (що забезпечує миттєвий відгук) та передачу даних про штрих через WebSocket іншим учасникам проєкту (що дозволяє їм бачити малювання в реальному часі). Після завершення штриха система додає дію до історії (для можливості скасування/повторення) та позначає наявність незбережених змін [6]. Більш детально, діаграму діяльності можна розглянути в додатку В.

1.3 Порівняльний аналіз існуючих програмних аналогів

Для визначення місця та доцільності розробки нового програмного забезпечення редактора двовимірної графіки та анімації необхідно провести порівняльний аналіз існуючих на ринку рішень. Такий аналіз дозволяє виявити сильні та слабкі сторони аналогів, визначити незадоволені потреби користувачів та обґрунтувати вибір напрямку власної розробки. Для порівняння обрано шість програмних продуктів, які представляють різні категорії редакторів двовимірної графіки та анімації: універсальні графічні редактори, спеціалізовані анімаційні інструменти, веб-орієнтовані рішення та безкоштовне програмне забезпечення з відкритим кодом.

Adobe Photoshop є галузевим стандартом для роботи з растровою графікою, ретуші, колірної корекції та створення складних композицій. Він має потужний набір інструментів малювання та виділення, підтримує роботу з шарами, масками, фільтрами та має вбудовану тимчасову шкалу для створення покадрової анімації. Однак, як зазначають професійні аніматори, функціонал таймлайну в Photoshop тривалий час залишається без оновлень, має суттєві

недоліки в інтерфейсі та не отримує покращень від розробників. Крім того, Photoshop не підтримує колаборативну роботу в реальному часі [7]. Приклад інтерфейсу Adobe Photoshop наведено на рис. 1.4.

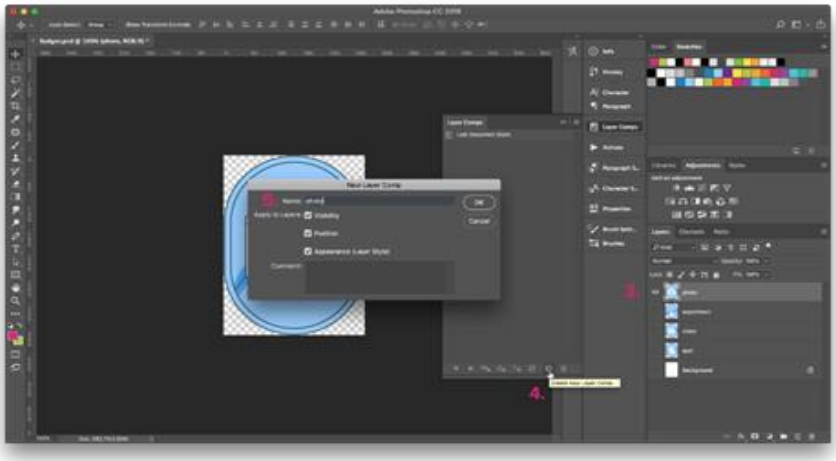


Рис. 1.4 – Приклад інтерфейсу Adobe Photoshop

Adobe Animate спеціалізується на створенні векторної та растрової анімації для вебу, телебачення та ігор. Програма підтримує автоматичне створення ключових кадрів та має потужні засоби для анімації персонажів. Однак, на думку користувачів, переміщення шарів і кадрів у Animate може бути складним та незручним [8]. Приклад створення анімації в Adobe Animate показано на рис. 1.5.

Рис. 1.5 – Створення анімації в Adobe Animate

Krita є безкоштовним програмним забезпеченням з відкритим вихідним кодом, що поєднує потужні засоби цифрового живопису з інструментами для покадрової анімації. Програма має професійний набір пензлів, підтримує

управління шарами та роботу з різноманітними файловими форматами. Krita активно розвивається завдяки спільноті розробників, однак не має вбудованих засобів для спільної роботи. Вигляд інтерфейсу Krita наведено на рис. 1.6.

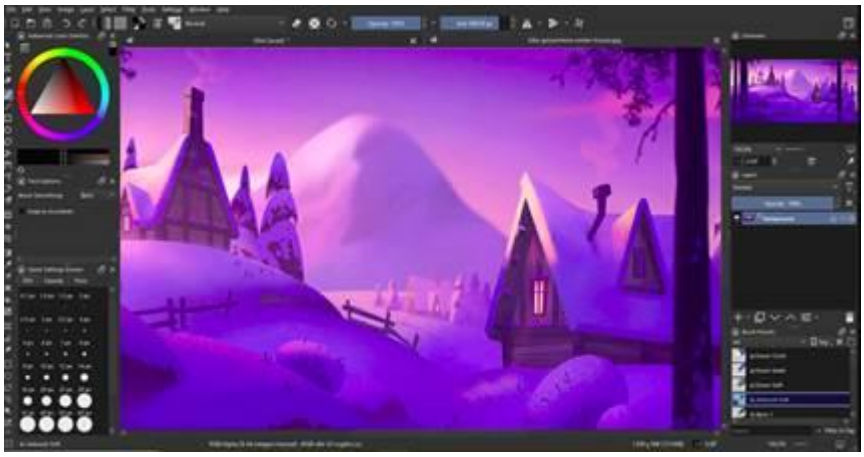


Рис. 1.6 – Вигляд інтерфейсу Krita

Aseprite є спеціалізованим редактором для створення піксельної графіки та анімації. Програма пропонує передові інструменти управління шарами, попередній перегляд анімації в реальному часі, потужну палітру кольорів та варіанти експорту. Однак, це десктопна програма без підтримки колаборації [10]. Приклад інтерфейсу Aseprite показано на рис. 1.7.

Рис. 1.7 – Aseprite підтримує можливість завантаження кастомного UI

Pencil2D – це безкоштовний відкритий редактор, орієнтований на створення традиційної покадрової анімації. Програма має простий та інтуїтивно зрозумілий інтерфейс, що робить її ідеальним вибором для початківців. Pencil2D підтримує як растрову, так і векторну графіку, але має обмежений функціонал

порівняно з професійними інструментами та не підтримує спільної роботи [11]. Мінімалістичний інтерфейс Pencil2D наведено на рис. 1.8.

Рис. 1.8 – Мінімалістичний інтерфейс Pencil2D

Photopea є веб-орієнтованим графічним редактором, який працює безпосередньо в браузері без необхідності встановлення. Він підтримує роботу з шарами, стилями, масками та форматами PSD, XCF, Sketch. Перевагою є доступність з будь-якого пристрою, однак Photopea створений переважно для редагування зображень, а не для анімації, і не має колаборативної підтримки [12]. Приклад веб-інтерфейсу Photopea показано на рис. 1.9.

Рис. 1.9 – Photopea вважається повноцінною альтернативою фотошопу

Зведене порівняння існуючих програмних аналогів подано в табл. 1.1.

Таблиця 1.1 – Порівняльний аналіз існуючих аналогів

Критерій	Adobe Photoshop	Adobe Animate	Krita	Aseprite	Pencil2D	Photopea
Тип ліцензії	Пропрієтарна (підписка)	Пропрієтарна (підписка)	Вільна (GPL)	Умовно-безкоштовна ($19.99)	Вільна (GPL)	Умовно-безкоштовна (з рекламою)
Платформа	Windows, macOS	Windows, macOS	Windows, macOS, Linux	Windows, macOS, Linux	Windows, macOS, Linux	Веб (будь-яка)
Растрова графіка	Так	Частково	Так	Так	Так	Так
Векторна графіка	Частково	Так	Частково	Ні	Так	Частково
Покадрова анімація	Так (обмежено)	Так	Так	Так	Так	Ні
Таймлайн	Так	Так	Так	Так	Так	Ні
Шари	Так	Так	Так	Так	Так	Так
Колаборація	Ні	Ні	Ні	Ні	Ні	Ні
Веб-доступ	Ні	Ні	Ні	Ні	Ні	Так

Проведений аналіз існуючих програмних аналогів показав, що сучасні редактори двовимірної графіки та анімації мають значний функціональний потенціал, проте більшість із них характеризуються складністю використання, високими системними вимогами або відсутністю можливостей спільної роботи

користувачів у режимі реального часу. Професійні системи, такі як Adobe Animate, орієнтовані переважно на професійних користувачів та мають складний інтерфейс. Водночас простіші редактори, зокрема Pencil2D, не забезпечують достатнього рівня функціональності та колаборативних можливостей [9]. Таким чином, розробка програмного забезпечення редактора двовимірної графіки та анімації є актуальною та доцільною, оскільки система орієнтована на поєднання функціональності, доступності та зручності колективної роботи користувачів.

1.4 Постановка завдання

На основі проведеного аналізу предметної області, порівняльного аналізу існуючих аналогів та сформульованих вимог до системи, необхідно визначити конкретне завдання на розробку програмного забезпечення редактора двовимірної графіки та анімації. Метою даної кваліфікаційної роботи є розробка програмного забезпечення для створення та редагування двовимірної графіки та покадрової анімації, яке забезпечує можливість спільної роботи кількох користувачів у реальному часі, зберігання проєктів на сервері, експорт результатів у поширені формати та інтуїтивно зрозумілий інтерфейс. Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Провести аналіз існуючих програмних рішень для створення двовимірної графіки та анімації, виявити їх переваги та недоліки, обґрунтувати вибір технологічного стеку для реалізації власного програмного забезпечення.

2. Розробити архітектуру веб-застосунку, що включає:

– серверну частину,

– клієнтську частину,

– базу даних для зберігання інформації про користувачів, проєкти, кадри, шари, коментарі та учасників,

3. Реалізувати систему керування користувачами, що забезпечує:

– реєстрацію та автентифікацію за електронною поштою,

– редагування профілю користувача.

4. Розробити функціонал керування проєктами анімації, що включає:

– створення, перейменування та видалення проєктів,

– налаштування параметрів проєкту (назва, розміри полотна, частота кадрів),

– розмежування прав доступу на основі ролей (власник, редактор, глядач),

– систему запрошень користувачів до проєкту за електронною поштою з унікальними токенами.

5. Реалізувати графічний редактор з такими інструментами та функціями:

– малювання пензликом з можливістю зміни розміру, прозорості та розмиття,

– гумка для стирання намальованого,

– заливка замкнених областей,

– створення геометричних фігур,

– піпетка для вибору кольору з зображення,

– виділення областей з можливістю переміщення, масштабування, копіювання, вирізання та вставки,

– робота з шарами,

– таймлайн для керування кадрами,

– відтворення анімації з заданою частотою кадрів, паузою, зупинкою та зацикленням,

– масштабування та панорамування полотна, повноекранний режим.

6. Розробити підсистему колаборативного редагування:

– відображення присутності інших користувачів у проєкті,

– трансляцію позицій курсорів з аватарами та іменами співавторів,

– передачу процесу малювання в реальному часі,

– блокування шарів для запобігання конфліктам,

– синхронізацію змін кадрів та шарів,

– систему коментарів з прив'язкою до проєкту або окремого кадру та миттєвими сповіщеннями.

7. Реалізувати збереження та завантаження даних:

– збереження кожного кадру,

– контроль версій для кадрів та шарів,

– автоматичне збереження кожні 30 секунд,

– генерацію попереднього перегляду для кожного кадру.

8. Розробити модуль експорту анімації

9. Провести тестування розробленого програмного забезпечення.

10. Підготувати інсталяційний пакет програмного забезпечення.

Поставлені завдання повинні бути вирішені з використанням сучасних технологій веб-розробки, що забезпечують високу продуктивність, безпеку та масштабованість системи. Результатом виконання роботи має стати повнофункціональний веб-орієнтований редактор двовимірної графіки та анімації, готовий до використання у навчальному процесі, для створення невеликих анімаційних проєктів або як основа для подальшого комерційного розвитку.

1.5 Формування функціональних та нефункціональних вимог до системи

Формування вимог до програмного забезпечення є одним із ключових етапів розробки системи, оскільки саме вимоги визначають функціональні можливості програмного продукту, його поведінку, продуктивність та особливості взаємодії з користувачем. На основі аналізу предметної області та програмного коду редактора двовимірної графіки та анімації було сформовано функціональні та нефункціональні вимоги до системи. Розроблювана система призначена для створення, редагування та відтворення двовимірної графіки й анімації із підтримкою колаборативних функцій та взаємодії користувачів у режимі реального часу.

Функціональні вимоги визначають набір функцій та сервісів, які повинна забезпечувати програмна система.

Вимоги до керування користувачами. Система повинна забезпечувати:

- реєстрацію користувачів;
- авторизацію та автентифікацію;
- збереження персональних даних користувача;
- керування профілем користувача.

Вимоги до роботи з проєктами. Система повинна забезпечувати:

- створення нового проєкту;
- редагування параметрів проєкту;
- відкриття існуючого проєкту;
- збереження стану проєкту;
- видалення проєкту;
- керування учасниками проєкту.

Вимоги до роботи з кадрами анімації. Система повинна забезпечувати:

- створення кадрів;
- редагування кадрів;
- зміну порядку кадрів;
- копіювання кадрів;
- видалення кадрів;
- перемикання між кадрами;
- відтворення анімації.

Вимоги до роботи із шарами. Система повинна забезпечувати:

- створення шарів;
- редагування шарів;
- зміни порядку шарів;
- приховування та блокування шарів;
- видалення шарів.

Вимоги до інструментів редагування графіки. Система повинна забезпечувати:

- малювання на полотні;
- зміну кольору та параметрів інструментів;
- масштабування робочої області;
- переміщення по полотну;
- редагування графічних елементів.

Вимоги до функцій колаборації. Система повинна забезпечувати:

- одночасну роботу кількох користувачів;
- синхронізацію змін у режимі реального часу;
- відображення активних користувачів;
- коментування проєкту;
- механізми уникнення конфліктів редагування.

Вимоги до експорту анімації. Система повинна забезпечувати:

- експорт анімації;
- підтримку кількох форматів файлів;
- формування готового анімаційного результату.

Нефункціональні вимоги визначають характеристики якості програмного забезпечення та умови його функціонування.

Вимоги до продуктивності. Система повинна:

- забезпечувати швидке оновлення графічного інтерфейсу;
- підтримувати роботу з кількома кадрами та шарами;
- забезпечувати мінімальні затримки під час collaborative-редагування;
- підтримувати realtime-синхронізацію змін.

Вимоги до надійності. Система повинна:

- забезпечувати стабільне збереження даних;
- запобігати втраті інформації;
- забезпечувати коректну обробку помилок;
- підтримувати механізми відновлення стану системи.

Вимоги до зручності використання. Система повинна:

- мати зрозумілий графічний інтерфейс;
- забезпечувати швидкий доступ до основних функцій;

- підтримувати інтуїтивне керування інструментами;
- забезпечувати зручну навігацію між кадрами та шарами.

Вимоги до масштабованості. Система повинна:

- підтримувати розширення функціональних можливостей;
- забезпечувати модульну структуру програмного забезпечення;
- підтримувати додавання нових інструментів та компонентів.

Вимоги до сумісності. Система повинна:

- працювати у сучасних веббраузерах;
- підтримувати мережеву взаємодію;
- забезпечувати коректну роботу на різних операційних системах.

Вимоги до безпеки. Система повинна:

- забезпечувати автентифікацію користувачів;
- підтримувати контроль доступу до проєктів;
- запобігати несанкціонованому редагуванню даних;
- забезпечувати захист користувацької інформації.

Вимоги до супроводу та модифікації. Система повинна:

- мати модульну архітектуру;
- забезпечувати можливість подальшого вдосконалення;
- підтримувати оновлення окремих компонентів без повної зміни системи.

У результаті аналізу предметної області та програмного коду було сформовано функціональні та нефункціональні вимоги до редактора двовимірної графіки та анімації. Визначені вимоги охоплюють основні аспекти роботи системи, включаючи створення та редагування графіки, взаємодію користувачів, експорт анімації, забезпечення продуктивності, надійності та безпеки програмного забезпечення. Сформований перелік вимог є основою для подальшого проєктування та реалізації програмної системи.

1.6 Висновки до розділу

У першому розділі було розглянуто предметну область створення та редагування двовимірної графіки й анімації. Під час аналізу було визначено, що ця сфера охоплює не лише інструменти малювання, а й роботу з кадрами, шарами, таймлайном, форматами збереження та експортом готового результату. Окрему увагу було приділено растровій і векторній графіці, а також основним підходам до створення анімації, зокрема покадровій анімації та анімації за ключовими кадрами.

Також у розділі було виконано моделювання предметної області за допомогою UML-діаграм. Це дало змогу визначити основних учасників системи, їхні ролі та сценарії взаємодії з майбутнім програмним забезпеченням. Було встановлено, що ключовими користувачами системи є власник проєкту, співавтор та неавторизований користувач.

Порівняльний аналіз існуючих програмних аналогів показав, що більшість сучасних редакторів мають достатньо широкий функціонал, однак часто не підтримують повноцінну спільну роботу в реальному часі або вимагають встановлення на локальний комп’ютер. Саме це підтвердило доцільність розробки веб-орієнтованого редактора двовимірної графіки та анімації з підтримкою колаборативних функцій. На основі проведеного аналізу було сформульовано постановку задачі, функціональні та нефункціональні вимоги до системи.

2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Проєктування логічної моделі даних

Для забезпечення персистентності інформації, цілісності даних, мінімізації надлишковості та оптимізації пошукових SQL-запитів у системі спільної розробки 2D-анімації необхідно спроєктувати логічну модель бази даних. Логічний рівень моделювання деталізує структуру сутностей, їхній атрибутивний склад, типи даних, а також фіксує бізнес-правила взаємодії через механізми первинних та зовнішніх ключів із визначенням кардинальності зв'язків. Розроблювана система базується на реляційній моделі даних, яка декомпонована на 11 взаємопов'язаних сутностей, що логічно охоплюють підсистеми автентифікації, менеджменту проєктів, графічного редагування, WebSocket-присутності користувачів та реалтайм-блокувань конфліктів.

Аналіз предметної області дозволив виділити наступні основні сутності логічної моделі:

1. User. Зберігає інформацію про зареєстрованого користувача системи. Містить атрибути: ідентифікатор, електронна пошта, пароль, відображуване ім'я, аватар, дата створення, дата останнього входу, прапорці активності та прав. Електронна пошта є унікальною.
2. Profile. Доповнює користувача налаштуваннями інтерфейсу. Містить атрибути: ідентифікатор, посилання на користувача, тема інтерфейсу.
3. AnimationProject. Основний контейнер для анімаційної роботи. Містить атрибути: ідентифікатор, назва, опис, ширина та висота полотна, частота кадрів, дати створення та оновлення, посилання на користувача-власника.

4. ProjectMember. Визначає права доступу користувача до конкретного проєкту. Містить атрибути: ідентифікатор, посилання на проєкт, посилання на користувача, роль, дата приєднання, посилання на користувача, який запросив, прапорець активності. Унікальне обмеження: одна пара (проєкт, користувач).
5. ProjectInvite. Забезпечує механізм запрошення нових учасників. Містить атрибути: ідентифікатор, посилання на проєкт, електронна пошта запрошеного, роль, унікальний токен, дати створення та закінчення терміну дії, дата прийняття, посилання на користувача, який прийняв, посилання на користувача, який запросив, статус.
6. Frame. Представляє один кадр анімації. Містить атрибути: ідентифікатор, посилання на проєкт, номер кадру , JSON-вміст кадру, версія вмісту, зображення попереднього перегляду, дати створення та оновлення.
7. Layer. Зберігає окремий шар всередині кадру. Містить атрибути: ідентифікатор, посилання на кадр, порядковий номер, назва, прапорець видимості, прозорість (0-100), версія вмісту.
8. ProjectPresenceSession. Відстежує активних користувачів у проєкті в реальному часі. Містить атрибути: ідентифікатор, посилання на проєкт, посилання на користувача, назва каналу, посилання на поточний кадр, роль, час приєднання, час останньої активності, прапорець активності.
9. FrameLock. Запобігає одночасному редагуванню кадру кількома користувачами. Містить атрибути: ідентифікатор, посилання на проєкт, посилання на кадр, посилання на користувача, посилання на сесію присутності, час отримання блокування, час закінчення блокування.
10. LayerLock. Блокує окремий шар для редагування. Містить атрибути: ідентифікатор, посилання на проєкт, посилання на кадр, посилання на шар, посилання на користувача, посилання на сесію присутності,

час отримання блокування, час останнього heartbeat, час закінчення блокування.

11. ProjectComment. Дозволяє обговорювати проєкт або окремі кадри. Містить атрибути: ідентифікатор, посилання на проєкт, посилання на кадр, посилання на автора, текст коментаря, дати створення та оновлення, прапорець розв'язання, час розв'язання, посилання на користувача, який розв'язав.

Опис зв’язків:

- Один користувач може бути власником багатьох проєктів.
- Один проєкт містить багато кадрів.
- Один кадр містить багато шарів.
- Один користувач може бути учасником багатьох проєктів.
- Один проєкт може мати багатьох учасників.
- ProjectMember – це асоціативна сутність між User та AnimationProject з додатковим атрибутом "роль" (зв’язок "багато-до-багатьох" між User та AnimationProject).
- Один проєкт може мати багато запрошень.
- Один користувач може написати багато коментарів.
- Один проєкт може мати багато коментарів.
- Один кадр може мати багато коментарів, але коментар може не прив'язуватись до кадру.
- Один проєкт може мати багато активних сесій присутності.
- Один кадр може мати не більше одного активного блокування.
- Один шар може мати не більше одного активного блокування.

ER-діаграма для логічної моделі даних наведена в додатку Д.

Логічна модель даних спроєктована з урахуванням вимог реляційності. Всі сутності мають унікальний первинний ключ (id). Атрибути є атомарними – наприклад, content_json зберігає структуровані дані у текстовому полі (JSON), що в контексті NoSQL-підходу допускається для гнучкості, але в реляційному сенсі це порушує першу нормальну форму (1NF).

Всі інші атрибути задовольняють 1NF. Потенційні транзитивні залежності відсутні, оскільки всі неключові атрибути залежать тільки від первинного ключа. Наприклад, у сутності Layer атрибути order, name, visible, opacity залежать лише від id (або від frame_id, але frame_id є зовнішнім ключем, що посилається на первинний ключ Frame, що відповідає вимогам 2NF та 3NF). Жодних транзитивних залежностей не виявлено. Таким чином, за винятком використання JSON-поля (content_json), логічна модель даних перебуває у третій нормальній формі (3NF) [13]. Використання JSON є свідомим компромісом для зручності зберігання ієрархічних даних шарів, що в контексті веб-застосунку є прийнятним.

2.2 Вибір системи управління інформаційним забезпеченням

Після розроблення логічної моделі даних необхідно обрати систему управління інформаційним забезпеченням, яка забезпечить фізичне зберігання, цілісність, безпеку та ефективний доступ до даних. У даній роботі розглянуто два основних підходи до організації зберігання даних: використання файлової системи та використання системи керування базами даних. Файлова система є найпростішим способом зберігання даних, де інформація розміщується у файлах певного формату (наприклад, JSON, XML, CSV). До переваг файлового підходу належать простота реалізації, відсутність потреби в додатковому програмному забезпеченні та висока швидкість доступу до окремих файлів. Однак файлова система має суттєві недоліки: відсутність механізмів контролю цілісності даних, складність забезпечення узгодженості при паралельному доступі, обмежені можливості вибірки та пошуку даних, а також проблеми з масштабуванням. Система керування базами даних забезпечує структуроване зберігання даних з підтримкою транзакцій, контролем цілісності, потужними механізмами вибірки (мова SQL) та розмежуванням прав доступу. До недоліків можна віднести більшу складність налаштування та адміністрування, а також додаткові витрати на

апаратне забезпечення. Для розроблюваної системи, яка передбачає роботу з великою кількістю користувачів, проєктів, кадрів та шарів, необхідні транзакційність, цілісність даних та можливість виконання складних запитів. Тому обрано підхід на основі СКБД.

На основі аналізу логічної моделі даних (підрозділ 2.1) встановлено, що структура даних є реляційною – дані організовано у вигляді таблиць з фіксованою схемою, а зв'язки між ними реалізовано через зовнішні ключі. Винятком є поле content_json у сутності Frame, яке зберігає ієрархічні дані в форматі JSON, однак це не суперечить використанню реляційної СКБД, оскільки сучасні реляційні бази даних підтримують JSON-типи. Відповідно, обрано реляційну СКБД, оскільки вона найкраще відповідає логічній моделі даних та забезпечує:

- підтримку транзакцій ACID (атомарність, узгодженість, ізоляція, довговічність);
- цілісність даних через обмеження зовнішніх ключів та унікальності;
- потужну мову запитів SQL;
- індекси для прискорення вибірки даних;
- засоби резервного копіювання та відновлення.

Для реалізації фізичної структури бази даних обрано дві системи залежно від середовища виконання:

PostgreSQL – обрано як основну СКБД для виробничого середовища. Вибір обґрунтовується наступними перевагами:

- Висока надійність та відповідність стандартам SQL;
- Нативна підтримка JSON та JSONB типів даних, що дозволяє ефективно зберігати та індексувати поле content_json кадрів [14];
- Підтримка паралельних запитів та висока продуктивність при великих навантаженнях;
- Безкоштовна ліцензія з відкритим вихідним кодом (PostgreSQL License);
- Активна спільнота та велика кількість інструментів для адміністрування.

- SQLite – обрано для середовища розробки та запуску модульних тестів. Вибір обґрунтовується:
- Відсутністю потреби в окремому серверному процесі (вбудована БД);
- Простотою налаштування та використання;
- Малій вазі та високій швидкості для невеликих обсягів даних;
- Можливістю зберігати базу даних у вигляді одного файлу (test.sqlite3), що зручно для тестування.

Окрім основної СКБД для зберігання персистентних даних, система потребує засобів для забезпечення WebSocket-з'єднань та колаборативної взаємодії в реальному часі. Для цієї мети обрано Redis – високопродуктивне сховище даних у пам'яті, яке використовується як бекенд каналів для Django Channels. Redis забезпечує:

- Швидку передачу повідомлень між різними екземплярами застосунку при горизонтальному масштабуванні;
- Підтримку патерну "pub/sub" (публікація/підписка) для розсилки WebSocket-подій;
- Мінімальні затримки (мікросекунди), що критично для колаборативного редагування.

У виробничому середовищі PostgreSQL та Redis можуть розміщуватися як на окремому сервері, так і в контейнерах Docker разом з основним застосунком. Для спрощення розгортання розроблено Docker Compose конфігурацію (файл docker-compose.dev.yml), яка дозволяє запустити всі компоненти системи (Django-застосунок, PostgreSQL, Redis) в ізольованих контейнерах [15].

На основі логічної моделі (ER-діаграми) та обраної СКБД PostgreSQL створено фізичну структуру бази даних, яка реалізована за допомогою Django ORM. Кожна сутність логічної моделі відповідає класу-моделі в коді, атрибути – полям моделі, а зв'язки – зовнішнім ключам або зв'язкам "багато-до-багатьох". Фізична структура бази даних генерується автоматично за допомогою механізму міграцій Django. Це забезпечує узгодженість між логічною моделлю, кодом

застосунку та фізичною БД, а також спрощує внесення змін у структуру даних. Детальний опис фізичної структури наведено у підрозділі 2.4.

2.3 Розробка структури інформаційної бази

У процесі створення програмного забезпечення редактора двовимірної графіки та анімації особливу увагу приділено розробці структури інформаційної бази, оскільки саме вона є основою збереження, обробки та взаємодії з даними. Ретельно спроектована структура бази даних забезпечує логічну цілісність, ефективність виконання запитів та можливість масштабування у майбутньому. Для реалізації інформаційної бази обрано реляційну СУБД PostgreSQL (для виробничого середовища) та SQLite (для розробки та тестування). Взаємодія з базою даних здійснюється через об'єктно-реляційний маппер (ORM) фреймворку Django, який дозволяє описувати структуру даних у вигляді Python-класів (моделей) та автоматично генерувати відповідні SQL-таблиці, індекси та обмеження [16].

2.3.1 Налаштування підключення до бази даних

Підключення до бази даних здійснюється через конфігураційний файл animstudio/settings.py. У виробничому середовищі використовується PostgreSQL, параметри підключення до якого задаються через змінні середовища для забезпечення безпеки. Код налаштування та створення таблиць БД можна переглянути в додатку Е.

2.3.2 Створення таблиць

Створення таблиць у реляційній базі даних відбувається через визначення класів-моделей у файлах models.py. Кожен клас відповідає окремій таблиці, а кожен атрибут класу – стовпцю таблиці. Для забезпечення ефективного зберігання даних необхідно визначати правильну структуру моделей, що відповідає вимогам бізнес-логіки системи.

Модель користувача (User) є розширенням стандартної моделі Django AbstractUser. Вона використовує електронну пошту як основний ідентифікатор замість імені користувача. Рядок username = None видаляє стандартне поле username, яке не використовується. Поле email є унікальним (unique=True) і використовується для входу (USERNAME_FIELD = "email"). Поле display_name зберігає відображуване ім'я користувача (наприклад, для показу в інтерфейсі). Поле avatar зберігає шлях до завантаженого зображення аватара. Поле created_at автоматично встановлюється при створенні запису (default=timezone.now, editable=False).

Модель профілю (Profile) розширює користувача додатковими налаштуваннями, пов'язаними з інтерфейсом. Внутрішній клас ThemePreference визначає можливі варіанти теми інтерфейсу (системна, світла, темна). Поле user є зв'язком "один до одного" з моделлю користувача. Це означає, що одному користувачеві відповідає рівно один профіль. Параметр on_delete=models.CASCADE вказує, що при видаленні користувача автоматично видаляється і його профіль. Параметр related_name="profile" дозволяє звертатися до профілю через user.profile [17].

Модель проєкту анімації (AnimationProject) є основним контейнером для анімаційної роботи. Вона зберігає параметри полотна, частоту кадрів та зв'язок з власником. Поле owner є зовнішнім ключем до моделі користувача. Один користувач може бути власником багатьох проєктів. Параметр on_delete=models.CASCADE означає, що при видаленні користувача всі його проєкти також будуть видалені. Поля width та height визначають розміри полотна в пікселях (за замовчуванням 1280×720). Поле fps визначає частоту кадрів анімації (за замовчуванням 12 кадрів на секунду). Поле created_at автоматично встановлюється при створенні (auto_now_add=True), а updated_at оновлюється при кожному збереженні (auto_now=True).

Модель учасника проєкту (ProjectMember) визначає права доступу користувача до конкретного проєкту. Вона реалізує зв'язок "багато-до-багатьох" між користувачами та проєктами з додатковим атрибутом role. Внутрішній клас

Role визначає три можливі ролі: власник, редактор, глядач. Поля project та user є зовнішніми ключами до відповідних таблиць. Поле invited_by зберігає інформацію про те, хто запросив користувача. При видаленні користувача, який запросив, це поле встановлюється в NULL. Клас Meta визначає метадані моделі. Зокрема, UniqueConstraint забезпечує унікальність пари (проєкт, користувач), тобто один користувач не може бути доданий до одного проєкту двічі.

Модель кадру (Frame) представляє один кадр анімації. Вона зберігає вміст кадру у форматі JSON, версію вмісту та зображення попереднього перегляду. Поле project зв'язує кадр з батьківським проєктом. Поле index визначає порядковий номер кадру в межах проєкту (починаючи з 1). Поле content_json зберігає JSON-рядок, що містить масив шарів із зображеннями в форматі base64. Поле content_revision використовується для контролю версій і запобігання конфліктам при одночасному збереженні. Поле preview_image зберігає попередній перегляд кадру. У класі Meta параметр ordering задає сортування кадрів за проєктом та номером. Параметр unique_together забезпечує унікальність пари (проєкт, номер кадру) – в одному проєкті не може бути двох кадрів з однаковим індексом.

Модель шару (Layer) представляє окремий шар всередині кадру. Поле frame зв'язує шар з батьківським кадром. Поле order визначає порядок накладання шарів (чим більше число, тим вище шар). Поле visible визначає, чи відображається шар на полотні. Поле opacity визначає прозорість шару у відсотках (0 – повністю прозорий, 100 – повністю непрозорий). Поле content_revision використовується для контролю версій вмісту шару.

Модель запрошення до проєкту (ProjectInvite) забезпечує механізм додавання нових учасників до проєкту за електронною поштою. Функція generate_project_invite_token генерує унікальний токен (випадковий рядок) для кожного запрошення. Функція default_project_invite_expiry встановлює термін дії запрошення – 7 днів від поточного моменту. Внутрішній клас Role визначає ролі, які можна призначити запрошеному (лише редактор або глядач; власник не може бути призначений через запрошення). Внутрішній клас Status визначає

можливі стани запрошення: очікує, прийнято, скасовано, закінчився. Поле token зберігає унікальний токен для доступу до сторінки прийняття запрошення. Поле expires_at визначає дату закінчення терміну дії запрошення. Поле invited_by зберігає користувача, який створив запрошення.

Для підвищення продуктивності виконання запитів до бази даних, особливо при зростанні обсягу даних, у моделях визначено додаткові індекси. Індекси прискорюють операції пошуку, фільтрації та сортування за певними полями. Приклад створення індексів наведено в моделі ProjectComment:

```
class ProjectComment(models.Model):
# ... поля моделі ...
class Meta:
        ordering = ['created_at', 'id']
        indexes = [
        models.Index(fields=['project', 'created_at'],
name='animation_p_project_59979d_idx'),
        models.Index(fields=['project', 'frame', 'created_at'],
name='animation_p_project_6b4b56_idx'),
        models.Index(fields=['project', 'is_resolved'],
name='animation_p_project_30b060_idx'),
        ]
```

Перший індекс прискорює пошук коментарів за проєктом з сортуванням за датою створення. Другий індекс прискорює пошук коментарів за проєктом та конкретним кадром. Третій індекс прискорює фільтрацію коментарів за статусом вирішення (is_resolved) в межах проєкту. Аналогічні індекси визначено для моделей ProjectPresenceSession, FrameLock та LayerLock для оптимізації запитів, пов'язаних з колаборативною взаємодією.

Django ORM дозволяє визначати обмеження на рівні бази даних, які гарантують цілісність даних. Приклад визначення унікального обмеження наведено в моделі ProjectMember:

```
class Meta:
```

```
constraints = [
    models.UniqueConstraint(fields=['project', 'user'],
name='unique_project_member'),
]
```

Django надає механізм сигналів, який дозволяє виконувати певні дії автоматично при настанні подій (наприклад, створення або збереження об'єкта). Цей механізм аналогічний SQL-тригерам, але реалізований на рівні застосунку.

```
from django.db.models.signals import post_save
from django.dispatch import receiver
from .models import Profile, User
@receiver(post_save, sender=User)
def ensure_profile_exists(sender, instance, created, **kwargs):
if created:
    Profile.objects.create(user=instance)
      return
  Profile.objects.get_or_create(user=instance)
```

Декоратор @receiver(post_save, sender=User) прив'язує функцію до сигналу post_save моделі User. Параметр created вказує, чи є цей запис новим (при створенні) або оновленням існуючого. При створенні нового користувача функція автоматично створює пов'язаний профіль. Якщо з якоїсь причини профіль відсутній, функція створює його при першому збереженні. Аналогічний сигнал визначено у файлі animation/signals.py для автоматичного додавання власника проєкту до списку учасників з роллю "власник":

```
from django.db.models.signals import post_save
from django.dispatch import receiver
from .models import AnimationProject
@receiver(post_save, sender=AnimationProject)
def ensure_project_owner_membership(sender, instance, raw, **kwargs):
if raw:
    return
```

instance.ensure_owner_membership()

Django автоматично генерує міграції – файли, які описують зміни в структурі бази даних (створення таблиць, додавання полів, створення індексів тощо). Міграції дозволяють зберігати історію змін та відтворювати структуру бази даних на різних середовищах (розробка, тестування, продакшен). Приклад автоматично згенерованої міграції для створення таблиці користувачів:

```
class Migration(migrations.Migration):
initial = True
dependencies = [
    ('auth', '0012_alter_user_first_name_max_length'),
]
operations = [
    migrations.CreateModel(
    name='User',
    fields=[
        ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
        ('password', models.CharField(max_length=128, verbose_name='password')),
        ('email', models.EmailField(max_length=254, unique=True, verbose_name='Email')),
        ('display_name', models.CharField(blank=True, max_length=150, verbose_name='Display name')),
        # ... інші поля ...
    ],
    ),
    # ... інші операції ...
]
```

Виконання міграцій здійснюється командою: «python manage.py migrate». Ця команда застосовує всі очікуючі міграції до бази даних, створюючи необхідні таблиці, індекси та обмеження.

У цьому підрозділі розглянуто процес розробки структури інформаційної бази для редактора двовимірної графіки та анімації. Описано створення таблиць через Django-моделі, визначення індексів для оптимізації запитів, обмежень для забезпечення цілісності даних, а також механізми автоматичного оновлення даних через сигнали. Використання Django ORM дозволило абстрагуватися від конкретної СУБД та зосередитися на логічній структурі даних.

2.4 Схема та фізична структура зберігання даних

Фізична структура бази даних є важливим елементом проєктування, який визначає зберігання, організацію та взаємозв’язки даних. У межах даної інформаційної системи структура бази даних визначається за допомогою Django ORM, який на основі Python-класів (моделей) автоматично генерує SQL-код для створення таблиць. Це дозволяє детально налаштувати атрибути таблиць, встановити обмеження цілісності даних (включаючи первинні та зовнішні ключі, унікальні обмеження), а також задати типи даних відповідно до специфіки збережуваної інформації. Загалом у базі даних реалізовано 11 основних таблиць, які описані нижче.

1. accounts_user – зберігає облікові записи користувачів. Містить унікальний ідентифікатор, електронну пошту (яка використовується для входу), хешований пароль, відображуване ім'я, шлях до файлу аватара, дату реєстрації, дату останнього входу, а також прапорці активності та прав доступу (адміністратор, персонал).

2. accounts_profile – зберігає додаткові налаштування користувача. Містить посилання на користувача та обрану тему інтерфейсу (системна, світла, темна).

3. animation_animationproject – зберігає проєкти анімації. Містить назву, опис, ширину та висоту полотна (за замовчуванням 1280×720 пікселів), частоту кадрів FPS (за замовчуванням 12), дати створення та оновлення, а також посилання на користувача-власника.

4. animation_projectmember – забезпечує розмежування прав доступу до проєкту. Містить посилання на проєкт та на користувача, роль учасника (власник, редактор або глядач), дату приєднання, посилання на користувача, який запросив, та прапорець активності. Унікальне обмеження на пару (проєкт, користувач) гарантує, що один користувач не може бути доданий до одного проєкту двічі.

5. animation_projectinvite – зберігає запрошення до проєкту. Містить посилання на проєкт, електронну пошту запрошеного, роль, яка призначається (редактор або глядач), унікальний токен для доступу, дати створення та закінчення терміну дії (за замовчуванням 7 днів), статус запрошення (очікує, прийнято, скасовано, закінчився), а також посилання на користувача, який створив запрошення, та користувача, який його прийняв.

6. animation_frame – зберігає кадри анімації. Містить посилання на проєкт, порядковий номер кадру, вміст кадру у форматі JSON (рядок, що містить масив шарів із зображеннями в форматі base64), версію вмісту (для контролю конфліктів при одночасному збереженні), шлях до зображення попереднього перегляду, а також дати створення та оновлення. Унікальне обмеження на пару (проєкт, номер кадру) гарантує, що в межах одного проєкту не може бути двох кадрів з однаковим номером.

7. animation_layer – зберігає шари, що належать до певного кадру. Містить посилання на кадр, порядковий номер шару (визначає порядок накладання), назву, прапорець видимості, прозорість (від 0 до 100 відсотків) та версію вмісту шару. При видаленні кадру всі його шари видаляються автоматично.

8. animation_projectpresencesession – відстежує активних користувачів у проєкті для колаборативної взаємодії. Містить посилання на проєкт та на користувача, унікальну назву WebSocket-каналу, посилання на поточний кадр

користувача, роль у проєкті, час підключення, час останньої активності та прапорець активності. Сесія вважається активною, якщо час останньої активності менше ніж 90 секунд тому.

9. animation_framelock – блокує кадри для запобігання одночасному редагуванню кількома користувачами. Містить посилання на проєкт, посилання на кадр (зв'язок "один до одного"), посилання на користувача, який заблокував, посилання на сесію присутності, час отримання блокування, час останнього heartbeat та час закінчення блокування (TTL становить 35 секунд). Heartbeat, який надсилається кожні 12 секунд, подовжує блокування.

10. animation_layerlock – блокує окремі шари для редагування. Містить аналогічні поля та обмеження, що й таблиця блокування кадрів, але замість кадру посилається на шар (зв'язок "один до одного"). Це дозволяє кільком користувачам редагувати різні шари одного кадру одночасно, але не дозволяє редагувати один шар двом користувачам.

11. animation_projectcomment – зберігає коментарі користувачів до проєкту загалом або до конкретного кадру. Містить посилання на проєкт, необов'язкове посилання на кадр (якщо коментар стосується конкретного кадру), посилання на автора, текст коментаря, дати створення та оновлення, прапорець розв'язання, дату розв'язання та посилання на користувача, який розв'язав коментар. При видаленні кадру коментарі, що належали йому, встановлюють посилання на кадр у NULL.

Кожна таблиця має первинний ключ (поле id), який забезпечує унікальність записів. Для встановлення взаємозв’язків між таблицями використовуються зовнішні ключі (ForeignKey). Наприклад, таблиця animation_frame має зовнішній ключ project_id, який посилається на первинний ключ таблиці animation_animationproject, що забезпечує зв’язок між кадрами та проєктом. Аналогічно, таблиця animation_layer має зовнішній ключ frame_id, який посилається на таблицю animation_frame, а таблиці animation_framelock та animation_layerlock посилаються на відповідні сесії присутності. Для коректного представлення даних у базі використовуються такі типи:

- BIGINT для унікальних ідентифікаторів та числових значень;
- VARCHAR(n) для рядків фіксованої довжини (електронна пошта, назви, токени);
- TEXT для великих текстових полів (JSON-вміст кадру, опис проєкту, тіло коментаря);
- BOOLEAN для прапорців (видимість шару, активність користувача);
- TIMESTAMP WITH TIME ZONE для зберігання дати та часу з урахуванням часового поясу;
- IMAGE для зберігання шляхів до файлів аватарів та попереднього перегляду кадрів.

Отже, фізична структура бази даних є логічно впорядкованою, гнучкою та масштабованою. Вона враховує специфіку зберігання анімаційних даних (кадри, шари, JSON-вміст), забезпечує надійне зберігання інформації про користувачів, проєкти та колаборативну взаємодію, а також підтримує важливі механізми забезпечення цілісності через зовнішні ключі та унікальні обмеження. Такий підхід дозволяє легко розширювати систему у майбутньому, додаючи нові сутності або функціональність, без шкоди для стабільності або якості даних.

2.5 Висновки до розділу

У другому розділі було виконано проєктування інформаційного забезпечення програмної системи. На основі вимог, визначених у першому розділі, було виділено основні сутності, необхідні для роботи редактора: користувач, профіль, проєкт анімації, учасник проєкту, запрошення, кадр, шар, коментар, сесія присутності, блокування кадру та блокування шару. Для кожної сутності було визначено основні атрибути та зв’язки з іншими елементами системи.

У процесі проєктування було створено логічну модель даних, яка відображає структуру майбутньої бази даних і забезпечує збереження інформації

про користувачів, проєкти, кадри, шари та колаборативну взаємодію. Модель побудовано з урахуванням вимог реляційності, цілісності даних і можливості подальшого розширення системи.

Для реалізації інформаційного забезпечення було обґрунтовано вибір реляційної системи керування базами даних PostgreSQL як основного рішення для виробничого середовища. Для розробки та тестування передбачено використання SQLite. Окремо було розглянуто застосування Redis для підтримки WebSocket-з’єднань і передачі подій у режимі реального часу. У результаті було сформовано фізичну структуру зберігання даних, яка відповідає логіці роботи системи та дозволяє забезпечити надійне збереження інформації.

3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ РЕДАКТОРА ДВОВИМІРНОЇ ГРАФІКИ ТА АНІМАЦІЇ

3.1 Визначення базових сутностей предметної області

На етапі розробки програмного забезпечення важливим кроком є визначення базових сутностей предметної області, які безпосередньо реалізуються у вигляді класів програми. Ці сутності відображають ключові об'єкти реального світу в контексті редактора двовимірної графіки та анімації та визначають структуру даних, які обробляються системою.

Сутність «Користувач» (User). Ця сутність представляє зареєстрованого користувача системи. Вона містить ідентифікаційні дані, персональну інформацію, а також службові поля, що визначають статус облікового.

Сутність «Профіль користувача» (Profile). Ця сутність доповнює користувача додатковими налаштуваннями, що не відносяться до автентифікації. Вона зберігає індивідуальні параметри інтерфейсу, зокрема обрану тему оформлення.

Сутність «Проєкт анімації» (AnimationProject). Це центральна сутність системи, яка представляє окремий анімаційний проєкт. Вона містить загальні параметри: назву, опис, розміри полотна, частоту кадрів, а також службові поля для відстеження дати створення та останнього оновлення.

Сутність «Учасник проєкту» (ProjectMember). Ця сутність реалізує зв'язок між користувачами та проєктами, визначаючи права доступу кожного учасника. Вона містить посилання на проєкт та на користувача, а також роль учасника, яка може бути однією з трьох: власник, редактор або глядач. Крім того, сутність зберігає дату приєднання, інформацію про те, хто запросив користувача, та прапорець активності.

Сутність «Запрошення до проєкту» (ProjectInvite). Ця сутність забезпечує механізм запрошення нових учасників до проєкту. Вона містить електронну пошту запрошеного, роль, яка призначається, унікальний токен для доступу, дати створення та закінчення терміну дії, а також статус запрошення. Запрошення створюється власником проєкту та може бути прийняте користувачем із відповідною електронною поштою.

Сутність «Кадр» (Frame). Ця сутність представляє окремий кадр анімації в межах проєкту. Вона містить порядковий номер кадру, вміст кадру у форматі JSON, версію вмісту для контролю конфліктів при одночасному збереженні, зображення попереднього перегляду, а також дати створення та оновлення.

Сутність «Шар» (Layer). Ця сутність представляє окремий шар всередині кадру. Вона містить порядковий номер, який визначає z-індекс, назву, прапорець видимості, прозорість та версію вмісту шару.

Сутність «Сесія присутності» (ProjectPresenceSession). Ця сутність відстежує активних користувачів у проєкті для забезпечення колаборативної взаємодії в реальному часі. Вона містить унікальну назву WebSocket-каналу, посилання на поточний кадр користувача, роль у проєкті, час підключення, час останньої активності та прапорець активності. Сесія вважається активною, якщо час останньої активності менше ніж 90 секунд тому.

Сутність «Блокування кадру» (FrameLock). Ця сутність реалізує механізм блокування кадрів для запобігання одночасному редагуванню кількома користувачами. Вона містить посилання на користувача, який заблокував кадр, посилання на сесію присутності, час отримання блокування, час останньої активності та час закінчення блокування.

Сутність «Блокування шару» (LayerLock). Ця сутність аналогічно реалізує блокування окремих шарів для редагування. Вона має взаємно однозначний зв'язок з шаром та містить ті самі поля, що й блокування кадру. Такий підхід дозволяє кільком користувачам одночасно редагувати різні шари одного кадру, але запобігає одночасному редагуванню одного шару.

Сутність «Коментар до проєкту» (ProjectComment). Ця сутність зберігає коментарі користувачів, які можуть відноситися до проєкту загалом або до конкретного кадру. Вона містить текст коментаря, дати створення та оновлення, прапорець розв'язання, дату розв'язання, а також посилання на автора та на користувача, який розв'язав коментар.

Визначені базові сутності є основою для побудови об'єктної моделі системи.

3.2 Моделювання структури та взаємодії об’єктів

Після визначення базових сутностей предметної області необхідно перейти до моделювання структури та взаємодії об’єктів програмної системи. Цей етап передбачає побудову діаграми класів UML, яка візуалізує статичну структуру системи: класи, їхні атрибути, методи та зв’язки між ними.

1. Клас User представляє зареєстрованого користувача системи. Він успадкований від AbstractUser Django, що забезпечує стандартні механізми автентифікації. Атрибути: email, display_name, avatar, created_at.

Методи:

- avatar_url() (property) – повертає URL аватара
- save() – перевизначений метод, який автоматично генерує display_name з електронної пошти, якщо його не задано

2. Клас Profile зберігає додаткові налаштування користувача. Атрибути: user, theme_preference.

3. Клас AnimationProject представляє анімаційний проєкт. Атрибути: owner, title, description, width, height, fps, created_at, updated_at.

Методи:

- ensure_owner_membership() – гарантує, що власник доданий до списку учасників з роллю OWNER

4. Клас ProjectMember визначає права доступу користувача до проєкту. Атрибути: project, user, role, joined_at, invited_by, is_active.

Методи:

- can_edit() – повертає True, якщо роль дозволяє редагування (OWNER або EDITOR)
- can_view() – повертає True, якщо роль дозволяє перегляд (будь-яка активна роль)
- can_manage_members() – повертає True, якщо роль дозволяє керувати учасниками (тільки OWNER)

5. Клас ProjectInvite представляє запрошення до проєкту. Атрибути: project, email, role, token, created_at, expires_at, accepted_at, accepted_by, invited_by, status.

Методи:

- is_expired() – повертає True, якщо термін дії минув
- is_pending() – повертає True, якщо статус PENDING і термін не минув
- can_be_accepted_by(user) – перевіряє, чи може користувач прийняти запрошення

6. Клас Frame представляє окремий кадр анімації. Атрибути: project, index, content_json, content_revision, preview_image, created_at, updated_at.

7. Клас Layer представляє шар всередині кадру. Атрибути: frame, order, name, visible, opacity, content_revision.

8. Клас ProjectPresenceSession відстежує активних користувачів у реальному часі. Атрибути: project (ForeignKey), user, channel_name, current_frame, role, joined_at, last_seen_at, is_active.

9. Клас FrameLock блокує кадр для редагування. Атрибути: project, frame, user, presence_session, acquired_at, last_heartbeat_at, expires_at.

10. Клас LayerLock блокує шар для редагування (аналогічний FrameLock, але для шарів).

11. Клас ProjectComment зберігає коментарі до проєкту або кадру. Атрибути: project, frame, author, body, created_at, updated_at, is_resolved, resolved_at, resolved_by.

12. Клас ProjectConsumer є WebSocket-контролером для колаборативної взаємодії. Атрибути: project_id, user_id, role, presence_session_id, owned_layer_lock_ids, project_group_name.

Методи:

- connect() – встановлення WebSocket-з'єднання
- disconnect() – закриття з'єднання
- receive_json() – обробка вхідних повідомлень
- send_event() – надсилання події клієнту
- activate_presence_session() – активація сесії присутності
- acquire_frame_lock(), acquire_layer_lock() – отримання блокувань
- release_frame_locks(), release_layer_locks() – зняття блокувань
- heartbeat_frame_lock(), heartbeat_layer_lock() – продовження блокувань

Діаграма класів UML візуалізує статичну структуру системи: класи, їхні атрибути, методи та зв’язки. Діаграма класів програми розташована у додатку Ж.

У цьому підрозділі на основі програмного коду визначено основні класи системи, їхні атрибути та методи. Розроблено діаграму класів, яка візуалізує статичну структуру редактора двовимірної графіки та анімації.

3.3 Організаційна структура програмного забезпечення

Організаційна структура програмного забезпечення представлена у вигляді діаграми пакетів, яка відображає логічний поділ системи редактора двовимірної графіки та анімації на окремі функціональні модулі (пакети) та взаємозв’язки між ними.

Уся система представлена у вигляді головного пакета, який охоплює п'ять основних підпакетів (кооперацій), кожен із яких відповідає за реалізацію окремого бізнес-процесу. Така модульна структура забезпечує зручність

супроводу, повторне використання компонентів та чіткий розподіл відповідальностей.

Пакет «accounts» (керування користувачами). Цей пакет є базовим і відповідає за реєстрацію, автентифікацію та керування обліковими записами користувачів. Він містить моделі даних (User, Profile), форми для реєстрації та редагування профілю, а також представлення (views) для входу, виходу та відновлення пароля. Пакет є незалежним і не потребує функціоналу інших пакетів, оскільки автентифікація є базовою потребою всієї системи.

Пакет «animation» (основна логіка анімації). Це центральний пакет системи, який містить всю бізнес-логіку роботи з анімаційними проєктами. Він включає підпакети:

- models – моделі даних (AnimationProject, Frame, Layer, ProjectMember, ProjectInvite, ProjectComment, ProjectPresenceSession, FrameLock, LayerLock)
- views – контролери для обробки HTTP-запитів (створення проєкту, редагування кадрів, експорт тощо)
- services – сервісні функції (invite_service, member_service)
- consumers – WebSocket-контролер для колаборативної взаємодії
- access – функції перевірки прав доступу
- locks – механізми блокування кадрів та шарів
- presence – відстеження присутності користувачів
- realtime – функції трансляції подій у реальному часі

Пакет «animation» залежить від пакета «accounts», оскільки для роботи з проєктами необхідна інформація про користувачів (власник, учасники).

Пакет «editor» (клієнтський редактор). Цей пакет реалізує фронтенд-частину графічного редактора, яка виконується в браузері користувача. Він включає:

- JavaScript-код для роботи з HTML5 Canvas (малювання, шари, таймлайн)
- WebSocket-клієнт для колаборативної взаємодії
- Модулі для експорту анімації

- Систему історії дій (undo/redo)

Пакет «editor» залежить від пакета «animation», оскільки він використовує API, надані серверною частиною, та обмінюється даними через WebSocket-з'єднання.

Пакет «animstudio» (конфігурація та маршрутизація). Цей пакет є кореневим і відповідає за загальну конфігурацію Django-проєкту. Він містить:

- settings.py – налаштування бази даних, встановлених додатків, middleware
- urls.py – маршрутизація HTTP-запитів
- asgi.py – налаштування ASGI для WebSocket-з'єднань
- routing.py – маршрутизація WebSocket-з'єднань

Пакет «animstudio» є збиральним і має залежності на всі інші пакети, оскільки саме в ньому визначається, які URL-адреси ведуть до яких представлень.

Пакет «static» (статичні файли). Цей пакет містить статичні ресурси, що використовуються клієнтською частиною:

- CSS-стилі для оформлення інтерфейсу
- JavaScript-файли (в тому числі editor.js)
- зображення (іконки інструментів, аватари за замовчуванням)

Пакет «static» залежить від пакета «editor» (оскільки надає йому ресурси) та використовується пакетом «animstudio» для обслуговування статичних файлів. Діаграма пакетів програми представлена в додатку И.

3.4 Компонентна структура системи

Компонентна структура програмного забезпечення редактора двовимірної графіки та анімації представлена у вигляді UML-діаграми компонентів. Вона відображає логічну архітектуру системи та демонструє взаємодію між основними програмними модулями. Архітектура системи побудована за клієнт-

серверним принципом та включає frontend-рівень, backend-рівень, підсистему realtime-взаємодії, модуль експорту анімації та систему зберігання даних. Такий підхід забезпечує модульність, масштабованість та зручність супроводу програмного забезпечення.

Далі розглянемо основні складові компонентної структури системи.

Компоненти користувацького інтерфейсу (Frontend Client). Frontend-частина системи реалізована за допомогою JavaScript-модулів та забезпечує взаємодію користувача із графічним редактором. До складу frontend-компонента входять:

- Canvas Editor – модуль роботи з полотном та графічними об’єктами;
- Timeline UI – компонент часової шкали анімації;
- Layer Manager – система керування шарами;
- Animation Preview – модуль попереднього перегляду анімації;
- Comment System UI – інтерфейс коментування;
- Collaboration UI – інтерфейс collaborative-взаємодії користувачів.

Дані компоненти забезпечують обробку дій користувача та передають запити до серверної частини через REST API та WebSocket-з’єднання.

Серверна частина системи (Django Backend). Backend-компонент реалізований на основі фреймворку Django та забезпечує обробку бізнес-логіки системи. До його складу входять:

- Authentication Module – модуль автентифікації та авторизації користувачів;
- Project Management Module – керування анімаційними проєктами;
- Frame & Layer Management – модулі роботи з кадрами та шарами;
- Comment Module – система коментарів;
- Invite & Membership Module – механізми запрошення користувачів та керування учасниками;
- Export Module – модуль експорту анімації;
- API Views – REST API для взаємодії frontend та backend;
- Access Control – система контролю доступу до проєктів.

Серверна частина виконує обробку запитів користувачів, взаємодіє з базою даних та координує роботу realtime-підсистеми.

Підсистема realtime-взаємодії (WebSocket / Realtime). Для забезпечення collaborative-функцій у системі реалізована окрема підсистема realtime-взаємодії на основі WebSocket. До її складу входять:

- ProjectConsumer – WebSocket-контролер обміну подіями;
- Presence Sessions – механізм відстеження активних користувачів;
- Frame Locks – система блокування кадрів;
- Layer Locks – система блокування шарів;
- Event Synchronization – механізм синхронізації змін між клієнтами.

Даний компонент забезпечує передачу подій у режимі реального часу та синхронізацію змін між кількома користувачами під час спільного редагування проєкту.

Компонент експорту анімації (Export Engine). Окремий компонент системи відповідає за формування та експорт готової анімації. До його складу входять:

- GIF Export – експорт анімації у формат GIF;
- MP4 Export – експорт у відеоформат MP4;
- WebM Export – експорт у формат WebM;
- FFmpeg Integration – інтеграція із бібліотекою FFmpeg для обробки відеоданих.

Компонент експорту отримує дані кадрів від backend-системи та формує кінцеві файли анімації.

Компонент зберігання даних (Database). Для збереження інформації система використовує централізовану базу даних, у якій зберігаються:

- користувачі;
- анімаційні проєкти;
- кадри;
- шари;
- коментарі;
- запрошення;

- сесії присутності;
- блокування кадрів та шарів.

Взаємодія серверної частини з базою даних реалізується через ORM-механізми Django.

Компонент файлового сховища (File Storage). Окремий компонент відповідає за зберігання файлів системи:

- аватарів користувачів;
- preview-зображень;
- експортованих файлів анімації;
- завантажених ресурсів проєкту.

Файлове сховище використовується backend-компонентом та модулем експорту. У компонентній структурі системи використовуються залежності та взаємодії між компонентами. Frontend-компонент взаємодіє із Django Backend через REST API та WebSocket-з’єднання. Серверна частина використовує базу даних для збереження інформації та взаємодіє з компонентом файлового сховища під час роботи з файлами. Realtime-підсистема забезпечує синхронізацію подій між користувачами та взаємодіє як із frontend-клієнтом, так і з backend-системою. Компонент експорту анімації отримує дані від backend-модуля та передає сформовані файли до файлового сховища. Діаграма компонентів представлена в додатку К.

Отже, компонентна структура системи забезпечує чітке розділення функціональності між окремими модулями програмного забезпечення. Такий підхід спрощує розробку, тестування, супровід та масштабування системи, а також дозволяє ефективно реалізувати колаборативні функції та механізми роботи з двомірною графікою та анімацією.

3.5 Обґрунтування вибору технологій та засобів розробки

Правильний вибір технологічного стеку є критично важливим етапом розробки програмного забезпечення, оскільки він безпосередньо впливає на продуктивність, надійність, масштабованість та зручність супроводу системи. Для реалізації редактора двовимірної графіки та анімації було обрано сучасний веб-стек, який забезпечує високу продуктивність клієнтської частини, потужну та безпечну серверну логіку, а також ефективну колаборативну взаємодію в реальному часі.

Для розробки серверної частини системи обрано мову програмування Python. Цей вибір обумовлений кількома факторами. По-перше, Python має простий та зрозумілий синтаксис, що прискорює процес розробки та полегшує підтримку коду. По-друге, Python має величезну екосистему бібліотек та фреймворків, зокрема для веб-розробки, роботи з базами даних, обробки зображень та відео. По-третє, Python є однією з найпопулярніших мов для створення веб-застосунків, що забезпечує велику спільноту розробників та велику кількість готових рішень. Для розробки клієнтської частини обрано мову JavaScript, оскільки вона є стандартом для веб-застосунків, працює безпосередньо в браузері та дозволяє створювати інтерактивні графічні інтерфейси з використанням HTML5 Canvas. Для реалізації серверної частини обрано фреймворк Django, який є одним із найпотужніших та найпопулярніших веб-фреймворків для Python. Django надає велику кількість вбудованих функцій, зокрема ORM для роботи з базами даних, систему маршрутизації URL, механізми автентифікації та авторизації, адміністративну панель, захист від CSRF-атак та XSS-вразливостей. Крім того, Django має детальну документацію та велику спільноту, що значно спрощує розробку та вирішення проблем, які виникають. Використання Django дозволило швидко створити надійну серверну основу з чіткою архітектурою MVT.

Для забезпечення колаборативної взаємодії в реальному часі обрано розширення Django Channels, яке додає підтримку асинхронних протоколів, зокрема WebSocket. Channels дозволяє обробляти довготривалі з'єднання, транслювати події групам користувачів та інтегрується з Redis як бекендом каналів. Завдяки Channels система отримала можливість передавати позиції курсорів, події малювання, блокування шарів та інші колаборативні дані в реальному часі без необхідності постійних HTTP-запитів. Для забезпечення роботи WebSocket-каналів у Django Channels обрано Redis – високопродуктивне сховище даних у пам'яті. Redis використовується як бекенд каналів для маршрутизації повідомлень між різними екземплярами застосунку при горизонтальному масштабуванні. Завдяки Redis система забезпечує мінімальні затримки передачі колаборативних даних.

Для обробки зображень та експорту анімації у формат GIF використовується бібліотека Pillow. Pillow дозволяє відкривати, маніпулювати та зберігати растрові зображення в різних форматах. У системі Pillow використовується для масштабування кадрів, накладання шарів з урахуванням прозорості та об'єднання послідовності зображень в анімований GIF-файл. Для експорту анімації у відеоформати MP4 та WebM використовується бібліотека FFmpeg – набір інструментів для обробки мультимедійних даних. FFmpeg викликається із застосунку як зовнішній процес для кодування послідовності зображень у відеофайл. Для MP4 використовується кодек H.264, який забезпечує високу якість стиснення та широку сумісність. Для WebM використовується кодек VP9, який підтримує прозорість (альфа-канал) та є оптимальним для веб-використання. Для реалізації автентифікації, зокрема входу через облікові записи Google, використовується бібліотека django-allauth. Вона забезпечує гнучку систему реєстрації, входу, відновлення пароля та соціальної автентифікації з підтримкою багатьох провайдерів OAuth. Використання allauth дозволило скоротити час розробки та забезпечити безпечну автентифікацію згідно з сучасними стандартами. Для інтеграції Django Channels з Redis використовується бібліотека channels-redis, яка реалізує необхідний інтерфейс

бекенду каналів. Вона забезпечує надійну доставку повідомлень між різними процесами та серверами, що є важливим для горизонтального масштабування системи. Для роботи з JWT під час генерації підписаних посилань на експортовані файли використовуються бібліотеки PyJWT та cryptography. Вони забезпечують безпечне кодування та декодування токенів з обмеженим терміном дії, що запобігає несанкціонованому доступу до файлів експорту. Для спрощення розгортання системи та забезпечення однакового середовища розробки, тестування та виробничого середовища використовуються Docker та Docker Compose. Docker дозволяє упакувати застосунок та всі його залежності (Django, PostgreSQL, Redis) в ізольовані контейнери. Docker Compose дозволяє описати та запустити багатоконтейнерний застосунок однією командою. Це значно спрощує процес інсталяції та налаштування системи на різних платформах.

На фронтенді використовується чистий JavaScript без додаткових фреймворків, оскільки робота з HTML5 Canvas вимагає безпосереднього доступу до API малювання, а надмірне використання абстракцій може призвести до зниження продуктивності. Вся логіка редактора реалізована в одному файлі editor.js, що дозволяє ефективно керувати станом, обробляти події миші та клавіатури, а також взаємодіяти з WebSocket. Такий підхід забезпечує високу швидкодію та плавність анімації.

Таким чином, обраний технологічний стек повністю відповідає вимогам до системи. Python та Django забезпечують надійну серверну частину, Django Channels та Redis – колаборативну взаємодію в реальному часі, PostgreSQL – надійне зберігання даних, а JavaScript та HTML5 Canvas – потужний та швидкий графічний редактор у браузері. Використання Docker спрощує розгортання системи, що робить її доступною для широкого кола користувачів.

3.6 Алгоритмізація та програмування програмних модулів

На етапі алгоритмізації здійснюється детальний опис внутрішньої логіки функціонування ключових модулів системи. Проєктування алгоритмів дозволяє формалізувати послідовність обчислювальних операцій, перевірок умов та циклів, що забезпечують стабільну роботу графічного редактора та підсистеми синхронізації в реальному часі. Нижче наведено специфікацію, програмну реалізацію та графічні блок-схеми базових алгоритмів системи.

1. Алгоритм заливки замкнених областей. Алгоритм заливки призначений для заповнення зв'язної області пікселів заданим кольором. Він використовується в інструменті «заливка» редактора. Вхідними даними алгоритму є координати точки, на якій користувач клацнув мишею, колір заливки та значення допуску, яке визначає, наскільки кольори можуть відрізнятися, щоб вважатися однаковими. Алгоритм працює наступним чином.

- Спочатку зчитується колір пікселя в точці клацання – цей колір називається цільовим.
- Далі створюється стек (або черга) для зберігання координат пікселів, які потрібно обробити.
- Початкова точка додається до стеку.
- Потім, поки стек не спорожніє, алгоритм виконує наступні кроки: витягує координати пікселя зі стеку, перевіряє, чи не виходить цей піксель за межі зображення, перевіряє, чи колір цього пікселя відрізняється від цільового не більше ніж на значення допуску, і якщо так – змінює колір пікселя на новий та додає до стеку чотирьох сусідніх пікселів (зверху, знизу, ліворуч, праворуч).

Для оптимізації алгоритм також використовує масив відвіданих пікселів, щоб не обробляти один піксель двічі. Це значно пришвидшує роботу на великих зображеннях. Блок-схему роботи алгоритму заливки замкнених областей наведено на рис. 3.1.

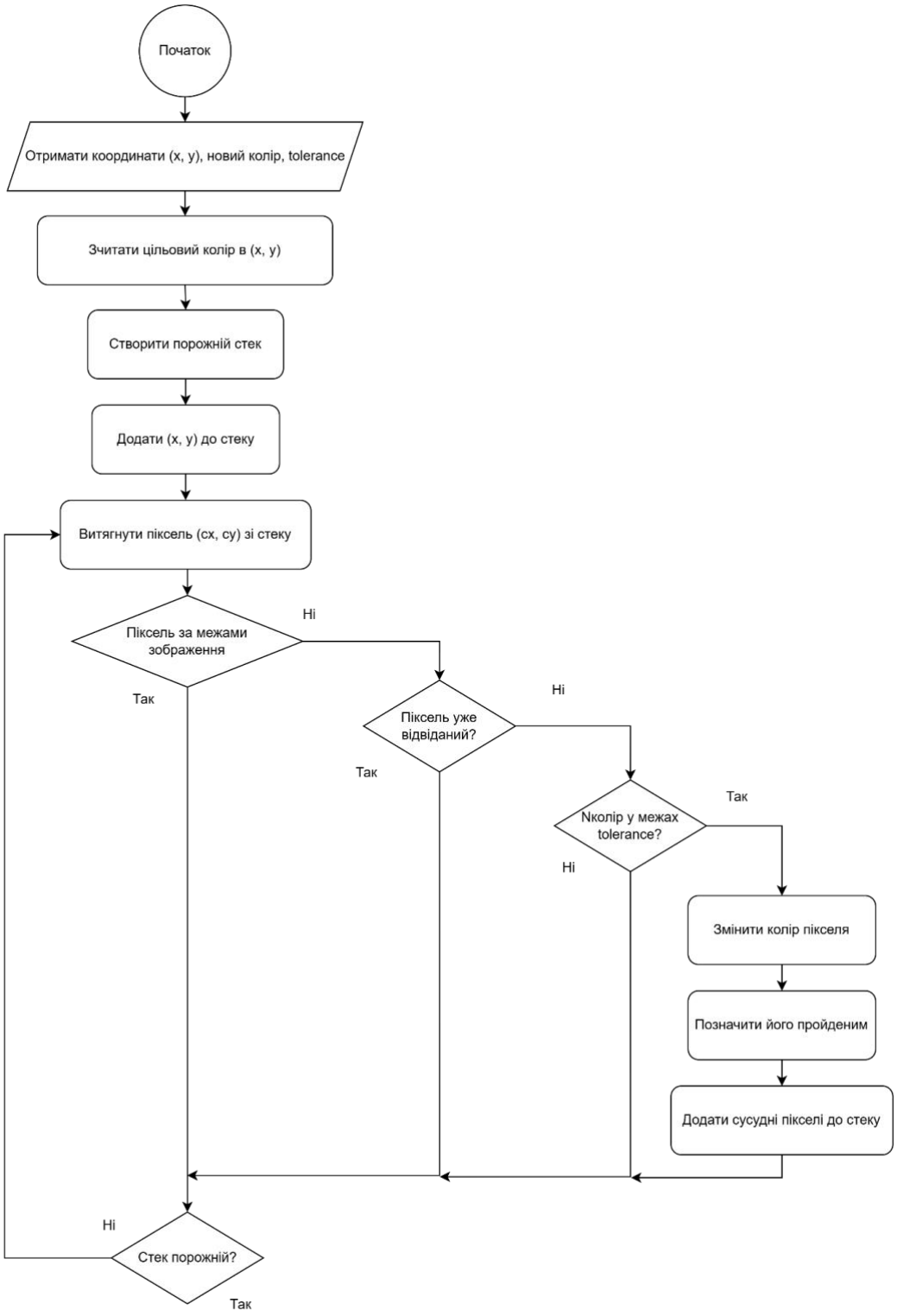


Рис. 3.1 – Блок-схема роботи алгоритму

Фрагмент програмного коду (JavaScript, editor.js):

```
function floodFill(startX, startY, options = {}) {
if (!bufferCtx || !bufferCanvas) return false;
```

```
        const width = bufferCanvas.width;
        const height = bufferCanvas.height;
        const x = Math.floor(startX);
        const y = Math.floor(startY);
        if (x < 0 || y < 0 || x >= width || y >= height) return false;
        const imageData = bufferCtx.getImageData(0, 0, width, height);
        const { data } = imageData;
        const startIndex = (y * width + x) * 4;
        const targetColor = [data[startIndex], data[startIndex + 1], data[startIndex + 2],
data[startIndex + 3]];
        const fillColor = hexToRgba(options.color || currentColor);
        if (colorsMatch(data, startIndex, fillColor)) return false;
        const visited = new Uint8Array(width * height);
        const stack = [x, y];
        while (stack.length > 0) {
                const currentY = stack.pop();
                const currentX = stack.pop();
                if (currentX === undefined || currentY === undefined) break;
                if (currentX < 0 || currentY < 0 || currentX >= width || currentY >= height)
continue;
                const offset = currentY * width + currentX;
                if (visited[offset]) continue;
              visited[offset] = 1;
                const pixelIndex = offset * 4;
                if (!colorsMatch(data, pixelIndex, targetColor)) continue;

              setPixelColor(data, pixelIndex, fillColor);
              stack.push(currentX + 1, currentY);
              stack.push(currentX - 1, currentY);
              stack.push(currentX, currentY + 1);
```

```
        stack.push(currentX, currentY - 1);
    }
      bufferCtx.putImageData(imageData, 0, 0);
    renderScene();
    return true;
    }
```

2. Алгоритм отримання блокування шару забезпечує колаборативне редагування, запобігаючи одночасній зміні одного шару кількома користувачами. Вхідними даними є ідентифікатори проєкту, кадру та шару, а також ідентифікатор користувача, сесії присутності та роль. Вихідними даними є статус операції (отримано або відмовлено) та інформація про блокування. Алгоритм працює в межах транзакції бази даних для забезпечення атомарності. Спочатку виконується очищення прострочених блокувань, які вже не є активними. Далі перевіряється, чи існує активна сесія присутності для користувача. Якщо сесія відсутня, алгоритм повертає відмову. Потім виконується пошук шару в базі даних. Якщо шар не знайдено, алгоритм повертає відмову. Далі перевіряється роль користувача – редагувати шар можуть лише власник проєкту або редактор. Якщо роль не дозволяє редагування, алгоритм повертає відмову. Після цього виконується пошук існуючого блокування для цього шару. Якщо блокування існує та належить іншій сесії, алгоритм повертає відмову з інформацією про те, хто заблокував шар. Якщо блокування існує та належить поточній сесії, алгоритм оновлює час закінчення блокування (heartbeat). Якщо блокування не існує, алгоритм створює нове. На завершення алгоритм знімає всі інші блокування, які належать поточній сесії (користувач може мати лише одне активне блокування одночасно), та повертає статус успіху.

Фрагмент програмного коду (Python, locks.py):

```
@transaction.atomic
def acquire_layer_lock(project_id, frame_id, layer_id, user_id, role, presence_session_id):
    now = timezone.now()
```

```
        stale_releases = cleanup_stale_layer_locks(project_id, now=now)

            presence_session = _get_active_presence_session(project_id, user_id,
presence_session_id, now)
        if presence_session is None:
                return {'status': 'denied', 'reason': 'invalid_session', 'lock': None, 'released':
stale_releases}

        layer = Layer.objects.select_related('frame').select_for_update().filter(
            pk=layer_id, frame_id=frame_id, frame__project_id=project_id
        ).first()
        if layer is None:
                return {'status': 'denied', 'reason': 'invalid_layer', 'lock': None, 'released':
stale_releases}

        if role not in {ProjectMember.Role.OWNER, ProjectMember.Role.EDITOR}:
                return {'status': 'denied', 'reason': 'read_only_role', 'lock': None, 'released':
stale_releases}

        existing_lock                                                    =
LayerLock.objects.select_for_update().filter(layer_id=layer.pk).first()
        if existing_lock and existing_lock.presence_session_id != presence_session.pk:
                return {'status': 'denied', 'reason': 'locked_by_other',
                    'lock': _serialize_layer_lock(existing_lock), 'released': stale_releases}

        expires_at = now + timedelta(seconds=LAYER_LOCK_TTL_SECONDS)
        if existing_lock is None:
                lock            =            LayerLock.objects.create(project_id=project_id,
frame_id=layer.frame_id, layer=layer,
                                        user_id=user_id, presence_session=presence_session,
```

```
                                last_heartbeat_at=now, expires_at=expires_at)
        else:
            existing_lock.user_id = user_id
            existing_lock.presence_session = presence_session
            existing_lock.last_heartbeat_at = now
            existing_lock.expires_at = expires_at
            existing_lock.save()
            lock = existing_lock

        other_locks = list(LayerLock.objects.filter(project_id=project_id, presence_session_id=presence_session.pk)
                           .exclude(layer_id=layer.pk))
        released = stale_releases + [_serialize_layer_lock(other_lock) for other_lock in other_locks]
        if other_locks:
            LayerLock.objects.filter(pk__in=[other_lock.pk for other_lock in other_locks]).delete()

        return {'status': 'acquired', 'reason': 'ok', 'lock': _serialize_layer_lock(lock), 'released': released}
```

3. Алгоритм збереження кадру відповідає за передачу даних поточного кадру від клієнта до сервера та оновлення бази даних. Вхідними даними є ідентифікатори проєкту та кадру, зображення попереднього перегляду в форматі base64, JSON-вміст кадру (структура шарів), ідентифікатор активного шару та версія шару. Алгоритм забезпечує контроль версій для запобігання конфліктам при одночасному збереженні. На клієнтській стороні алгоритм починається з перевірки, чи є незбережені зміни. Якщо змін немає, алгоритм завершується без дій. Потім формується payload, який містить зображення у форматі base64, JSON-вміст кадру, ідентифікатор активного шару та версію шару. Далі виконується HTTP-запит до сервера за відповідним URL. Сервер отримує запит, перевіряє

права доступу користувача (чи має він право редагувати проєкт) та валідує отримані дані. Потім сервер виконує пошук кадру в базі даних у межах транзакції. Якщо вказано активний шар, сервер перевіряє, чи існує блокування цього шару для поточної сесії користувача. Якщо блокування відсутнє, сервер повертає помилку. Також сервер перевіряє, чи збігається передана версія шару з поточною версією в базі даних. Якщо версії не збігаються, це означає, що інший користувач встиг змінити шар, тому сервер повертає помилку, а клієнт повинен оновити дані. Після успішної перевірки сервер оновлює JSON-вміст кадру, зберігає зображення попереднього перегляду, збільшує версії кадру та шару на одиницю, а потім транслює подію про зміну всім підключеним клієнтам через WebSocket.

Фрагмент програмного коду (Python, views.py):

```
@login_required
@require_POST
def frame_save(request, pk, index):
project = get_editable_project_or_404(request.user, pk)

try:
        payload = json.loads(request.body.decode('utf-8'))
except json.JSONDecodeError:
        return JsonResponse({'ok': False, 'error': 'Invalid JSON.'}, status=400)

frame = get_object_or_404(Frame, project=project, index=index)
active_layer_id = payload.get('active_layer_id')
   active_layer_revision = payload.get('active_layer_revision')
   presence_session_id = payload.get('presence_session_id')

with transaction.atomic():
      active_layer = None
        if active_layer_id is not None:
```

```
                active_layer = Layer.objects.select_for_update().filter(frame=frame, pk=active_layer_id).first()
                if active_layer is None:
                    return JsonResponse({'ok': False, 'error': 'Invalid active layer.'}, status=400)
                if not presence_session_holds_layer_lock(project.pk, active_layer.pk, request.user.pk, presence_session_id):
                    return JsonResponse({'ok': False, 'error': 'Layer lock required.'}, status=409)

                if active_layer_revision != active_layer.content_revision:
                    return JsonResponse({'ok': False, 'error': 'Layer content is stale. Refresh the frame and try again.'}, status=409)

                if payload.get('content_json') is not None:
                 merged_content = _merge_active_layer_content(frame.content_json, payload['content_json'], active_layer_id)
                 frame.content_json = merged_content

                if payload.get('image_data') is not None:
                decoded = base64.b64decode(payload['image_data'].split(',')[1])
                 frame.preview_image.save(f'frame_{index}.png', ContentFile(decoded), save=False)

                if active_layer is not None:
                 active_layer.content_revision += 1
                 active_layer.save()

            frame.content_revision += 1
            frame.save()
```

```
        project.save(update_fields=['updated_at'])
        broadcast_project_event(project.pk, 'frame_content_updated', {'frame_id': frame.pk})

    return JsonResponse({'ok': True, 'frame_revision': frame.content_revision})
```

Наведені алгоритми демонструють ключову логіку роботи редактора двовимірної графіки та анімації: обробку зображень на клієнтській стороні, колаборативну взаємодію через блокування шарів та надійне збереження даних з контролем версій. Всі алгоритми реалізовані з урахуванням вимог продуктивності та безпеки.

3.7 Розробка інтерфейсу користувача

Інтерфейс користувача редактора двовимірної графіки та анімації реалізований у вигляді веб-сторінки з використанням HTML, CSS та JavaScript. На початку роботи користувач має створити акаунт та залогінитися у відповідній формі. Сторінку логіну користувача показано на рис. 3.2.

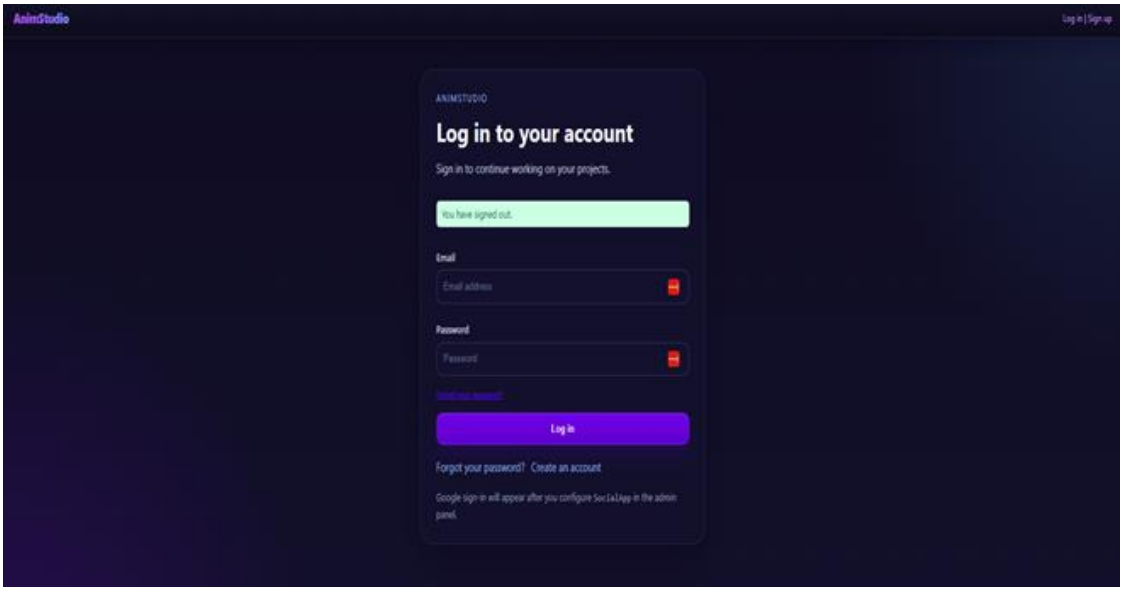


Рис. 3.2 – Сторінка логіну користувача

Потім користувачу пропонується створити новий проєкт або переглянути вже створені. Сторінку швидкого старту наведено на рис. 3.3. Вікно створення нового проєкту показано на рис. 3.4.

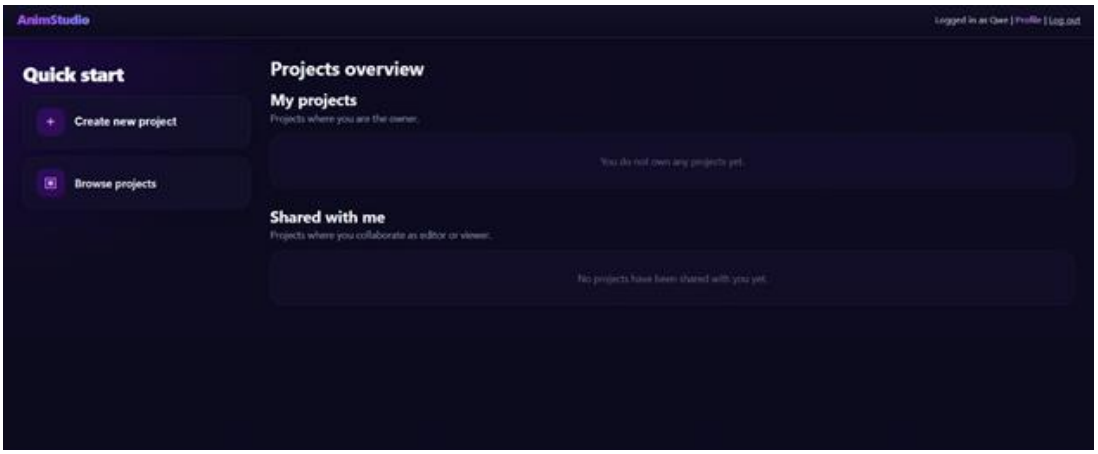


Рис. 3.3 – Сторінка швидкого старту

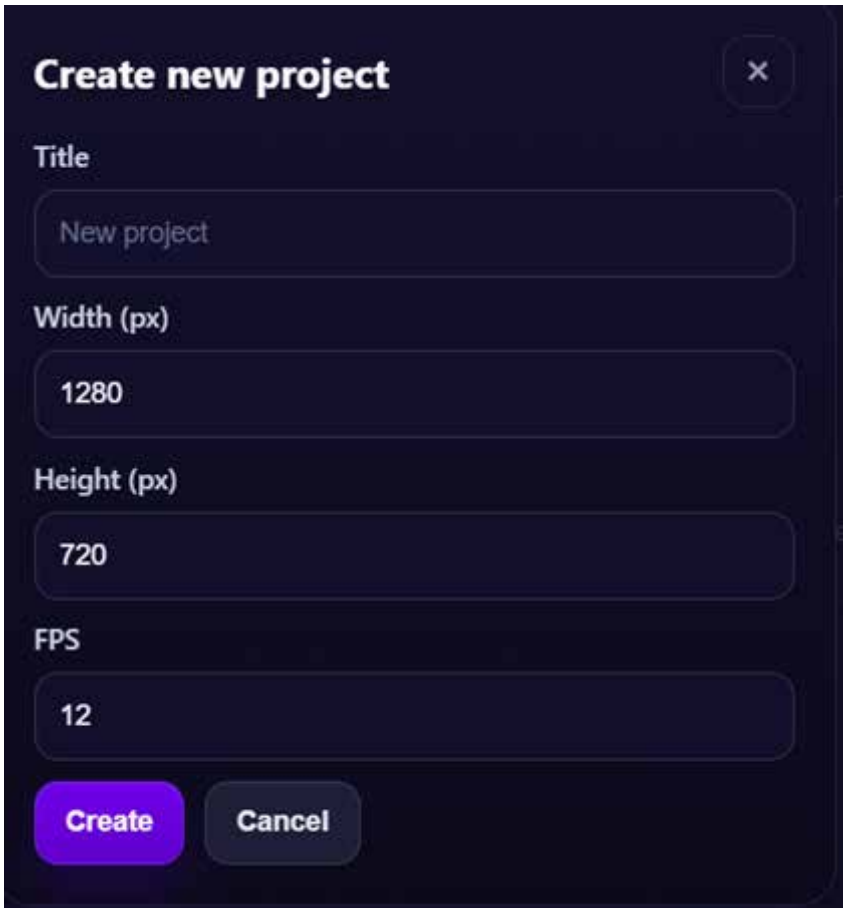


Рис. 3.4 – Вікно створення проєкту

Основним елементом є canvas-полотно, на якому відбувається малювання та відтворення анімації. Вигляд полотна анімації наведено на рис. 3.5.

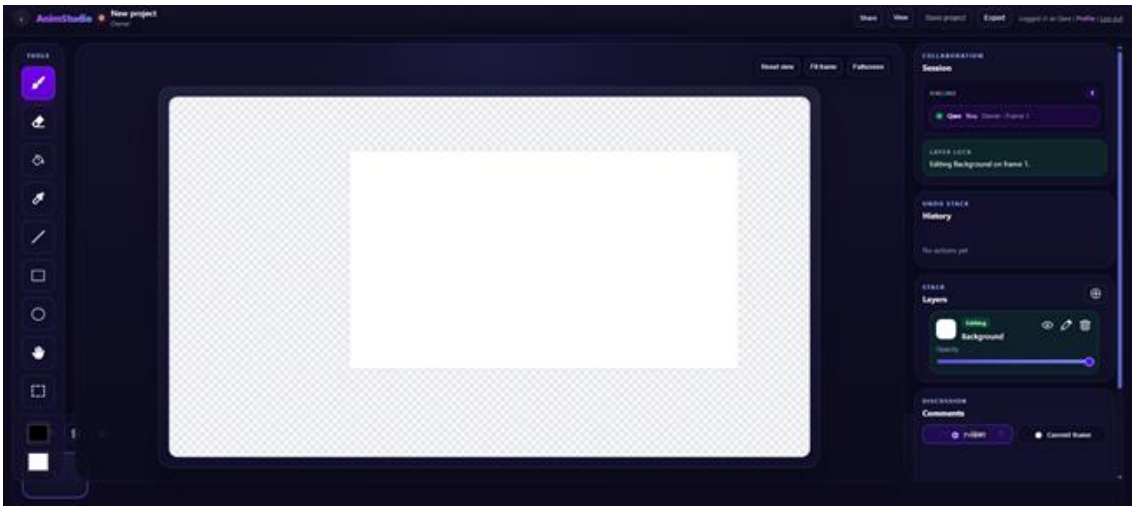


Рис. 3.5 – Полотно анімації

Інтерфейс складається з кількох функціональних зон: панель інструментів (пензлик, гумка, заливка, геометричні фігури, виділення, піпетка, панорамування), панель керування кольором, панель шарів (список шарів з можливістю зміни видимості, прозорості та порядку), таймлайн для керування кадрами (додавання, видалення, дублювання, зміна порядку, відтворення анімації), панель присутності (список користувачів онлайн) та панель коментарів. Панель інструментів редактора показано на рис. 3.6. Панелі колаборації, шарів та дискусії наведено на рис. 3.7.

Рис. 3.6 – Панель інструментів

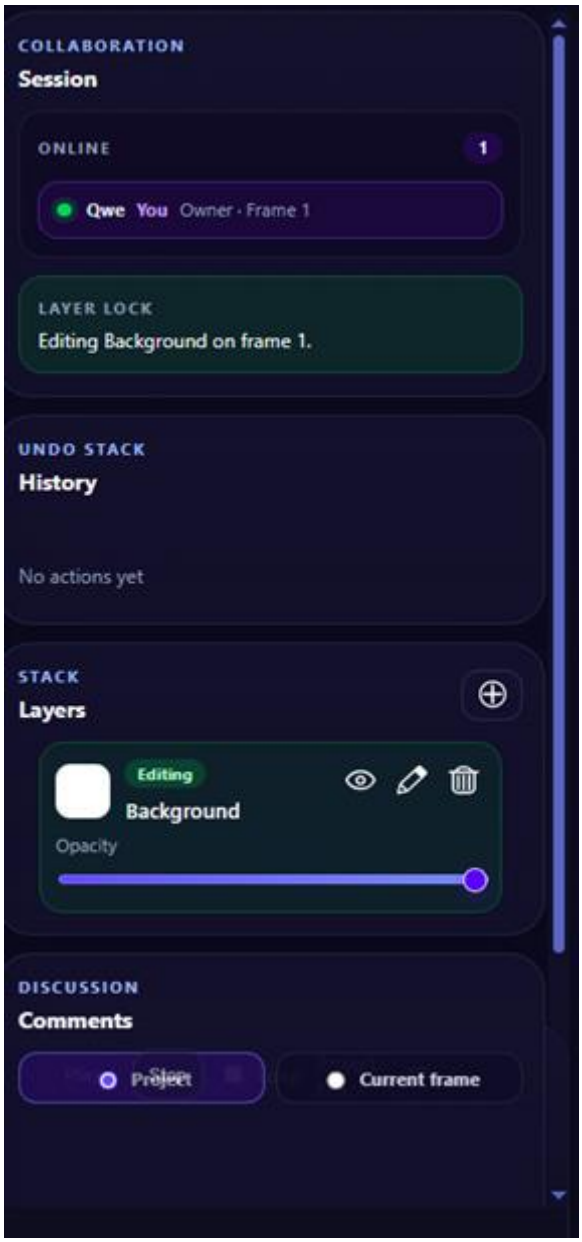


Рис. 3.7 – Панелі колаборації, шарів та дискусії

Усі елементи інтерфейсу підтримують світлу та темну теми оформлення.

3.8 Висновки до розділу

У третьому розділі було розглянуто безпосередню розробку програмного забезпечення редактора двовимірної графіки та анімації. Було визначено базові сутності предметної області, які реалізуються у вигляді класів програми. Серед них основними є користувач, проєкт, кадр, шар, учасник проєкту, запрошення,

коментар, а також класи, що відповідають за присутність користувачів і блокування елементів під час спільного редагування.

Було побудовано модель структури та взаємодії об’єктів, описано організаційну структуру програмного забезпечення та визначено основні пакети системи. Такий підхід дозволив логічно розділити відповідальність між окремими частинами програми: керуванням користувачами, роботою з анімаційними проєктами, клієнтським редактором, налаштуваннями Django-проєкту та статичними файлами.

У розділі також було обґрунтовано вибір технологій розробки. Для серверної частини використано Python і Django, для роботи в реальному часі — Django Channels і WebSocket, для зберігання даних — PostgreSQL, для кешування та передачі повідомлень — Redis, а для клієнтської частини — JavaScript і HTML5 Canvas. Окремо було описано алгоритми, що забезпечують роботу ключових функцій системи, зокрема заливку області, блокування шару та збереження кадру. У результаті було реалізовано веб-редактор із підтримкою малювання, роботи з шарами, кадрами, таймлайном, експортом і колаборативним режимом.

4. ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Тестування програмного забезпечення

Тестування розробленого програмного забезпечення проводилося з метою перевірки коректності роботи функціональних модулів, надійності системи, а також виявлення та усунення потенційних помилок. У процесі тестування було використано комбінацію методів: модульне тестування, інтеграційне тестування, тестування WebSocket-з'єднань та ручне функціональне тестування.

Модульне тестування. Для серверної частини системи було написано набір модульних тестів з використанням вбудованого в Django механізму тестування (unittest). Тести охоплюють основні моделі даних, форми, представлення та сервісні функції. Зокрема, протестовано створення користувачів та профілів, генерацію запрошень до проєктів, перевірку прав доступу на основі ролей, механізми блокування шарів та кадрів, а також функції експорту анімації. Всі тести розташовані у файлах accounts/tests.py та animation/tests.py. Інтеграційне тестування. Для перевірки взаємодії між компонентами системи було проведено інтеграційне тестування. Тестувалася взаємодія клієнтської частини з серверною через REST API, правильність обробки HTTP-запитів та відповіді сервера. Також тестувалася інтеграція з базою даних: створення, читання, оновлення та видалення записів через Django ORM, а також коректність роботи транзакцій. Тестування WebSocket-з'єднань. Оскільки ключовою особливістю системи є колаборативна взаємодія в реальному часі, було проведено окреме тестування WebSocket-комунікації. З використанням бібліотеки channels.testing та класу WebsocketCommunicator протестовано встановлення з'єднання, передачу повідомлень різних типів (присутність, блокування, малювання), а також коректне закриття з'єднання. Було перевірено, що система коректно обробляє

одночасне підключення кількох користувачів, транслює події всім учасникам проєкту та автоматично звільняє блокування при відключенні. Ручне функціональне тестування. Для перевірки роботи клієнтського графічного редактора було проведено ручне тестування. Тестувалися всі інструменти малювання (пензлик, гумка, заливка, фігури), робота з шарами (створення, видалення, зміна прозорості, порядку), керування кадрами на таймлайні, відтворення анімації, лукова шкіра, масштабування та панорамування полотна. Також перевірялася робота колаборативних функцій: відображення курсорів інших користувачів, трансляція малювання, коментарі та сповіщення.

Результати тестування. За результатами тестування всі основні функціональні вимоги до системи виконано. Модульні тести успішно проходять, WebSocket-з’єднання встановлюються та працюють коректно, а ручне тестування підтвердило зручність та інтуїтивність інтерфейсу. Виявлені в процесі тестування помилки були виправлені. Найбільш критичними виявилися помилки, пов’язані з синхронізацією блокувань шарів при одночасному редагуванні, які були усунені шляхом додавання додаткових перевірок версій та heartbeat-механізму. Загалом, система є стабільною та готовою до використання.

4.2 Вимоги до апаратного та програмного забезпечення

Для коректної роботи програмного забезпечення редактора двовимірної графіки та анімації необхідно забезпечити відповідність апаратного та програмного забезпечення комп'ютера або сервера мінімальним вимогам, наведеним нижче.

Для визначення умов експлуатації програмного забезпечення було побудовано діаграму розгортання системи, на якій показано основні вузли інфраструктури, компоненти застосунку та зв’язки між ними. Діаграму розгортання редактора двовимірної графіки та анімації наведено в додатку Л.

Система складається з робочої станції користувача, веб-/прикладного сервера, сервера бази даних PostgreSQL, сервера Redis, файлового сховища та зовнішнього сервісу Google OAuth. Користувач взаємодіє із системою через веббраузер за протоколами HTTPS та WSS. Серверна частина обробляє HTTP-запити, WebSocket-події, взаємодіє з базою даних через ORM/psycopg, використовує Redis як channel layer для realtime-обміну та зберігає медіафайли у файловому сховищі.

Вимоги до клієнтської частини. Оскільки система є веб-орієнтованою, користувачеві не потрібно встановлювати додаткове програмне забезпечення, крім сучасного веб-браузера. Мінімальні вимоги до апаратного забезпечення клієнта включають процесор із частотою не менше 1,5 ГГц, 2 ГБ оперативної пам'яті та екран із роздільною здатністю не менше 1280x720 пікселів. Рекомендовані вимоги: процесор 2 ГГц та більше, 4 ГБ оперативної пам'яті, екран із роздільною здатністю 1920x1080 пікселів. Програмне забезпечення клієнта повинно включати один із сучасних веб-браузерів: Google Chrome, Mozilla Firefox, Safari або Microsoft Edge. Браузер повинен підтримувати технологію HTML5 Canvas, WebSocket та JavaScript (ES6). Для роботи з системою необхідне стабільне підключення до інтернету зі швидкістю не менше 1 Мбіт/с; для комфортної колаборативної роботи рекомендується швидкість від 5 Мбіт/с.

Вимоги до серверної частини. Серверна частина системи може бути розгорнута на фізичному сервері, віртуальній машині або в контейнерах Docker. Мінімальні вимоги до апаратного забезпечення сервера: процесор із 2 ядрами та частотою 2 ГГц, 4 ГБ оперативної пам'яті, 20 ГБ вільного дискового простору. Рекомендовані вимоги: 4 ядра процесора, 8 ГБ оперативної пам'яті, 50 ГБ дискового простору (або більше, залежно від кількості проєктів та кадрів, що зберігаються). Для роботи застосунку необхідно встановити інтерпретатор Python версії 3.10 або новішої, менеджер пакетів pip та віртуальне середовище. Основними залежностями серверної частини є фреймворк Django, Django Channels для підтримки WebSocket, channels-redis для інтеграції з Redis, а також

бібліотеки psycopg2 для роботи з PostgreSQL, Pillow для обробки зображень та django-allauth для автентифікації. Як система керування базами даних використовується PostgreSQL версії 14 або новішої. Для забезпечення роботи WebSocket-каналів необхідно встановити Redis версії 6 або новішої, який використовується як бекенд каналів Django Channels. Для експорту анімації у відеоформати (MP4, WebM) на сервері повинен бути встановлений FFmpeg (версія 4.0 або новіша). Розгортання за допомогою Docker. Для спрощення розгортання системи у виробничому середовищі рекомендовано використовувати Docker та Docker Compose. У складі інсталяційного пакету надаються файли Dockerfile (для створення образу застосунку) та docker-compose.dev.yml (для запуску всіх необхідних сервісів: веб-застосунку, PostgreSQL та Redis). За допомогою команди docker-compose up можна розгорнути повнофункціональну систему на будь-якому сервері з встановленим Docker. Вимоги до мережі. Для роботи системи необхідно відкрити наступні порти на сервері: порт 80 (HTTP) або 443 (HTTPS) для доступу до веб-застосунку, порт 5432 для доступу до PostgreSQL (якщо база даних знаходиться на окремому сервері) та порт 6379 для доступу до Redis (якщо Redis знаходиться на окремому сервері). При використанні Docker всі внутрішні порти налаштовуються автоматично.

Дотримання зазначених вимог забезпечує стабільну та продуктивну роботу системи як для окремого користувача, так і для колаборативної взаємодії кількох користувачів одночасно.

4.3 Склад інсталяційного пакету

Інсталяційний пакет програмного забезпечення призначений для розгортання системи на сервері або локальному комп'ютері. Пакет постачається у вигляді Docker-образів, що забезпечує легкість встановлення, ізоляцію компонентів та відтворюваність середовища.

До складу інсталяційного пакету входять:

Код застосунку. Пакет містить усі файли вихідного коду системи, включаючи Python-модулі (Django-додатки accounts, animation, animstudio), JavaScript-файли (зокрема editor.js), HTML-шаблони, CSS-стилі та статичні файли (іконки, зображення).

Docker-конфігурація. Для контейнеризації системи надаються файл Dockerfile (описує побудову образу веб-застосунку) та файл docker-compose.dev.yml (визначає всі необхідні сервіси: веб-застосунок, базу даних PostgreSQL, Redis). За допомогою команди docker-compose up система запускається повністю автоматично.

Файли конфігурації. Пакет містить приклад файлу .env.example, в якому перелічено змінні середовища для налаштування системи: секретний ключ Django, параметри підключення до бази даних (PostgreSQL), параметри підключення до Redis, налаштування електронної пошти та параметри безпеки (SSL, HSTS). Для розгортання необхідно скопіювати цей файл у .env та заповнити значення змінних відповідно до середовища.

Список залежностей. Файл requirements.txt містить перелік усіх Python-бібліотек, необхідних для роботи системи, зокрема Django, Django Channels, channels-redis, psycopg2, Pillow, django-allauth, PyJWT та інші. Встановлення залежностей виконується автоматично під час побудови Docker-образу.

Інструкція з встановлення. Текстовий файл README.txt містить короткі інструкції щодо запуску системи за допомогою Docker, а також інструкції для ручного встановлення (без Docker) для розробників.

Таким чином, для встановлення системи користувачеві або адміністратору достатньо мати встановлені Docker та Docker Compose, завантажити інсталяційний пакет та виконати команду docker-compose up у кореневій папці проєкту [18]. Після цього система стає доступною за адресою http://localhost:8000. Для виробничого використання необхідно налаштувати змінні середовища у файлі .env (зокрема, згенерувати власний секретний ключ, налаштувати параметри бази даних Redis та налаштувати HTTPS).

4.4 Висновки до розділу

У четвертому розділі було розглянуто питання впровадження та експлуатації розробленого програмного забезпечення. Для перевірки працездатності системи було проведено тестування основних модулів, зокрема серверної частини, взаємодії з базою даних, WebSocket-з'єднань і функцій клієнтського графічного редактора. Результати тестування показали, що основні функціональні можливості системи працюють коректно, а виявлені під час перевірки помилки були усунені.

Також у розділі було визначено вимоги до апаратного та програмного забезпечення. Оскільки система є веб-орієнтованою, користувачеві достатньо мати сучасний браузер із підтримкою HTML5 Canvas, JavaScript і WebSocket. Для серверної частини визначено вимоги до процесора, оперативної пам'яті, дискового простору, а також необхідних програмних компонентів: Python, Django, PostgreSQL, Redis, FFmpeg, Docker і Docker Compose.

Окремо було описано склад інсталяційного пакету, до якого входять вихідний код застосунку, Docker-конфігурація, файл залежностей, приклад налаштувань середовища та інструкція зі встановлення. Це дає змогу розгорнути систему як локально, так і на сервері. Отже, розроблене програмне забезпечення є підготовленим до практичного використання, тестування та подальшого розвитку.

ВИСНОВКИ

У дипломній кваліфікаційній роботі на тему «Програмне забезпечення редактора двовимірної графіки та анімації» розроблено повнофункціональний веб-орієнтований графічний редактор із підтримкою покадрової анімації та колаборативного режиму реального часу. За результатами виконання роботи поставлена мета була досягнута повністю, а завдання вирішені в повному обсязі. Отримано такі основні наукові та практичні результати:

1. Проведено системний аналіз предметної області та виконано порівняльний аналіз існуючих аналогів. На основі виявлених переваг та недоліків існуючих рішень сформульовано комплекс функціональних та нефункциональних вимог до системи, що розробляється.
2. Спроєктовано архітектуру веб-застосунку на основі клієнт-серверної моделі з використанням асинхронного протоколу WebSocket для забезпечення взаємодії користувачів у режимі реального часу. Клієнтська частина реалізована на мові JavaScript з використанням технології HTML5 Canvas, а серверна – на базі фреймворку Django з інтеграцією Channels.
3. Розроблено логічну та фізичну структуру даних. Створено ER-діаграму бази даних, яка включає сутності: користувач, проєкт, кадр, шар, учасник, запрошення, коментар, сесія присутності, блокування кадру та блокування шару. Обґрунтовано вибір СУБД PostgreSQL та SQLite для збереження інформації, а також системи Redis як високопродуктивного бекенду каналів.
4. Програмно реалізовано повний набір інструментів графічного редактора: пензлик із підтримкою динамічної зміни розміру, прозорості та розмиття, гумка, заливка, геометричні фігури (лінія, прямокутник, еліпс). Реалізовано розвинену систему виділення (прямокутне,

еліптичне, ласо, чарівна паличка) з функціями переміщення, масштабування, копіювання, вирізання та вставки об'єктів.

5. Розроблено підсистему керування сценою та часом. Реалізовано роботу з шарами (створення, видалення, перейменування, зміна видимості та прозорості), інтерактивний таймлайн із можливістю керування відтворенням анімації, а також функцію «лукової шкіри» для візуального аналізу та перегляду суміжних кадрів. Забезпечено надійне збереження кадрів у форматі JSON із контролем версій та автоматичним збереженням стану кожні 30 секунд.
6. Впроваджено колаборативну складову через WebSocket-з’єднання, яка забезпечує: відображення присутності користувачів у проєкті, трансляцію позицій курсорів з аватарами співавторів, передачу процесу малювання в реальному часі, синхронізацію змін кадрів та шарів, систему коментування з миттєвими сповіщеннями. Для запобігання конфліктам інтегровано механізми динамічного блокування шарів та кадрів.
7. Реалізовано серверний модуль медіа-обробки, який забезпечує експорт готової анімації у формати PNG, GIF, MP4 та WebM з можливістю гнучкої зміни роздільної здатності та частоти кадрів.
8. Проведено комплексне тестування розробленого ПЗ, яке включало написання модульних та інтеграційних тестів для бекенду, спеціалізоване WebSocket-тестування на стійкість з'єднань, а також ручне функціональне тестування клієнтського фронтенду.

Проведений порівняльний аналіз показав суттєву перевагу розробленої системи «AnimStudio» над більшістю рішень завдяки наявності потужної вбудованої колаборативної підтримки без необхідності встановлення додаткового програмного забезпечення чи розширень на локальний комп'ютер користувача.

Перспективами подальшого розвитку розробленого програмного забезпечення є додавання підсистеми роботи зі звуковими доріжками, інтеграція

інструментів векторної графіки, реалізація алгоритмів автоматичного твінінгу (комп'ютерної інтерполяції кадрів), створення адаптивного мобільного додатку, а також подальше розширення переліку підтримуваних форматів імпорту та експорту медіафайлів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Adobe. Raster vs. vector: What are the differences? [Електронний ресурс]. – Режим доступу: https://www.adobe.com/creativecloud/file-types/image/comparison/raster-vs-vector.html (дата звернення: 22.05.2026).
2. Adobe. Frame-by-frame animation with Animate [Електронний ресурс]. – Режим доступу: https://helpx.adobe.com/animate/using/frame-by-frame-animation.html (дата звернення: 22.05.2026).
3. Adobe. The complete guide to image file formats [Електронний ресурс]. – Режим доступу: https://www.adobe.com/express/discover/image-file-guide (дата звернення: 22.05.2026).
4. Django Software Foundation. User authentication in Django [Електронний ресурс]. – Режим доступу: https://docs.djangoproject.com/en/6.0/topics/auth/ (дата звернення: 22.05.2026).
5. FFmpeg Developers. FFmpeg Documentation [Електронний ресурс]. – Режим доступу: https://ffmpeg.org/documentation.html (дата звернення: 22.05.2026).
6. IETF. RFC 6455: The WebSocket Protocol [Електронний ресурс]. – Режим доступу: https://www.rfc-editor.org/rfc/rfc6455 (дата звернення: 22.05.2026).
7. Martin R. Clean Architecture: A Craftsman’s Guide to Software Structure and Design. – Boston : Prentice Hall, 2017. – 432 ст. – Режим доступу: https://raw.githubusercontent.com/sdcuike/Clean-Code-Collection%20and%20Design.pdf (дата звернення: 13.04.2026).
8. Asynchronous Programming in Python [Електронний ресурс] // Real Python Team. – Режим доступу: https://realpython.com/async-io-python/ (дата звернення: 13.04.2026).
9. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. – Boston : Addison-Wesley, 1994. – 395 ст. – Режим доступу: https://www.javier8a.com/itc/bd1/articulo.pdf (дата звернення: 13.04.2026).

10. WebSocket with Django Channels: ASGI Setup & Deployment [Електронний ресурс]. – 2026. – Режим доступу: https://websocket.org/guides/frameworks/django/ (дата звернення: 16.04.2026).
11. IETF. RFC 6455: The WebSocket Protocol [Електронний ресурс] / I. Fette, A. Melnikov. – 2011. – Режим доступу: https://www.rfc-editor.org/rfc/rfc6455 (дата звернення: 16.04.2026).
12. WHATWG. HTML Living Standard: The canvas element [Електронний ресурс]. – Режим доступу: https://html.spec.whatwg.org/multipage/canvas.html (дата звернення: 16.04.2026).
13. W3C. HTML5: A vocabulary and associated APIs for HTML and XHTML (Canvas API) [Електронний ресурс] / I. Hickson, D. Hyatt. – 2008. – Режим доступу: https://www.w3.org/TR/2008/WD-html5-20080610/diff/the-canvas.html (дата звернення: 16.04.2026).
14. PostgreSQL Global Development Group. PostgreSQL 13 Documentation: Chapter 8.14. JSON Types [Електронний ресурс]. – 2025. – Режим доступу: https://www.postgresql.org/docs/13/datatype-json.html (дата звернення: 16.04.2026).
15. FFmpeg Developers. FFmpeg Documentation: Libavformat – Muxing and Demuxing Library [Електронний ресурс]. – Режим доступу: https://ffmpeg.org/libavformat.html (дата звернення: 16.04.2026).
16. Pillow Contributors. python-pillow/Pillow: 12.0.0 [Електронний ресурс] / Zenodo. – 2025. – Режим доступу: https://zenodo.org/records/17362416 (дата звернення: 16.04.2026).
17. Gentle J., Kleppmann M. Collaborative Text Editing with Eg-walker: Better, Faster, Smaller [Електронний ресурс] / J. Gentle, M. Kleppmann // arXiv preprint arXiv:2409.14252v1. – 2024. – Режим доступу: https://arxiv.org/abs/2409.14252 (дата звернення: 16.05.2026).
18. Docker Inc. Docker Documentation: Getting Started with Docker [Електронний ресурс]. – Режим доступу: https://docs.docker.com/compose/gettingstarted/ (дата звернення: 16.05.2026).

ДОДАТОК А

Фрагменти коду програми

Log_filters.py:

```
import logging
import re


DISALLOWED_HOST_PATTERN = re.compile(r"Invalid HTTP_HOST header: '([^']+)'")


class IgnoreScannerDisallowedHostFilter(logging.Filter):
    RESERVED_NOISE_HOSTS = {
        'example.com',
        'www.example.com',
    }

    def filter(self, record):
        message = record.getMessage()
        match = DISALLOWED_HOST_PATTERN.search(message)
        if not match:
            return True

        host = match.group(1).strip().lower().rstrip('.')
        if ':' in host:
            host = host.split(':', 1)[0]
        if host.endswith('.example.com'):
            return False
        return host not in self.RESERVED_NOISE_HOSTS
```

routing.py:

```
from django.urls import re_path

from animation.consumers import ProjectConsumer


websocket_urlpatterns = [
    re_path(r"^ws/projects/(?P<project_id>\d+)/$", ProjectConsumer.as_asgi()),
]
```

Asgi.py:

```
import os

from channels.auth import AuthMiddlewareStack
```

```
from channels.routing import ProtocolTypeRouter, URLRouter
from django.core.asgi import get_asgi_application

os.environ.setdefault("DJANGO_SETTINGS_MODULE", "animstudio.settings")

django_asgi_app = get_asgi_application()

from animstudio.routing import websocket_urlpatterns


application = ProtocolTypeRouter(
    {
        "http": django_asgi_app,
        "websocket": AuthMiddlewareStack(
            URLRouter(websocket_urlpatterns),
        ),
    }
)
```

Manage.py:

```
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault("DJANGO_SETTINGS_MODULE", "animstudio.settings")
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == "__main__":
    main()
```

models.py:

```
from datetime import timedelta
```

```
import uuid

from django.conf import settings
from django.db import models
from django.utils import timezone


def generate_project_invite_token():
    return uuid.uuid4().hex


def default_project_invite_expiry():
    return timezone.now() + timedelta(days=7)


class AnimationProject(models.Model):
    owner = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        on_delete=models.CASCADE,
        related_name='animation_projects',
    )
    title = models.CharField(max_length=200, verbose_name='Project title')
    description = models.TextField(blank=True, verbose_name='Description')
    width = models.PositiveIntegerField(default=1280, verbose_name='Canvas width
(px)')
    height = models.PositiveIntegerField(default=720, verbose_name='Canvas height
(px)')
    fps = models.PositiveIntegerField(default=12, verbose_name='Frames per second')
    created_at = models.DateTimeField(auto_now_add=True, verbose_name='Created at')
    updated_at = models.DateTimeField(auto_now=True, verbose_name='Updated at')
    # Audio fields can be added later if needed.

    def __str__(self):
        return self.title

    def ensure_owner_membership(self):
        ProjectMember.objects.update_or_create(
            project=self,
            user=self.owner,
            defaults={
                'role': ProjectMember.Role.OWNER,
                'invited_by': None,
                'is_active': True,
            },
        )
```

```
class ProjectMember(models.Model):
    class Role(models.TextChoices):
        OWNER = 'owner', 'Owner'
        EDITOR = 'editor', 'Editor'
        VIEWER = 'viewer', 'Viewer'

    project = models.ForeignKey(
        AnimationProject,
        on_delete=models.CASCADE,
        related_name='memberships',
    )
    user = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        on_delete=models.CASCADE,
        related_name='project_memberships',
    )
    role = models.CharField(
        max_length=16,
        choices=Role.choices,
        default=Role.VIEWER,
    )
    joined_at = models.DateTimeField(auto_now_add=True)
    invited_by = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        null=True,
        blank=True,
        on_delete=models.SET_NULL,
        related_name='sent_project_memberships',
    )
    is_active = models.BooleanField(default=True)

    class Meta:
        constraints = [
            models.UniqueConstraint(
                fields=['project', 'user'],
                name='unique_project_member',
            ),
        ]

    def __str__(self):
        return f'{self.user} in {self.project} ({self.role})'

    def can_edit(self):
        return self.is_active and self.role in {self.Role.OWNER, self.Role.EDITOR}
```

```
    def can_view(self):
        return self.is_active and self.role in set(self.Role.values)

    def can_manage_members(self):
        return self.is_active and self.role == self.Role.OWNER


class ProjectInvite(models.Model):
    class Role(models.TextChoices):
        EDITOR = 'editor', 'Editor'
        VIEWER = 'viewer', 'Viewer'

    class Status(models.TextChoices):
        PENDING = 'pending', 'Pending'
        ACCEPTED = 'accepted', 'Accepted'
        REVOKED = 'revoked', 'Revoked'
        EXPIRED = 'expired', 'Expired'

    project = models.ForeignKey(
        AnimationProject,
        on_delete=models.CASCADE,
        related_name='invites',
    )
    email = models.EmailField()
    role = models.CharField(
        max_length=16,
        choices=Role.choices,
        default=Role.VIEWER,
    )
    token = models.CharField(
        max_length=255,
        unique=True,
        default=generate_project_invite_token,
        editable=False,
    )
    created_at = models.DateTimeField(auto_now_add=True)
    expires_at = models.DateTimeField(default=default_project_invite_expiry)
    accepted_at = models.DateTimeField(null=True, blank=True)
    accepted_by = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        null=True,
        blank=True,
        on_delete=models.SET_NULL,
        related_name='accepted_project_invites',
    )
```

```
    invited_by = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        on_delete=models.CASCADE,
        related_name='sent_project_invites',
    )
    status = models.CharField(
        max_length=16,
        choices=Status.choices,
        default=Status.PENDING,
    )

    def __str__(self):
        return f'{self.email} -> {self.project} ({self.role})'

    def is_expired(self):
        return timezone.now() >= self.expires_at

    def is_pending(self):
        return self.status == self.Status.PENDING and not self.is_expired()

    def can_be_accepted_by(self, user):
        if not user or not getattr(user, 'is_authenticated', False):
            return False
        if not getattr(user, 'email', ''):
            return False
        return self.is_pending() and user.email.casefold() == self.email.casefold()


class ProjectComment(models.Model):
    project = models.ForeignKey(
        AnimationProject,
        on_delete=models.CASCADE,
        related_name='comments',
    )
    frame = models.ForeignKey(
        'Frame',
        null=True,
        blank=True,
        on_delete=models.SET_NULL,
        related_name='comments',
    )
    author = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        on_delete=models.CASCADE,
        related_name='project_comments',
```

```
    )
    body = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    is_resolved = models.BooleanField(default=False)
    resolved_at = models.DateTimeField(null=True, blank=True)
    resolved_by = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        null=True,
        blank=True,
        on_delete=models.SET_NULL,
        related_name='resolved_project_comments',
    )

    class Meta:
        ordering = ['created_at', 'id']
        indexes = [
            models.Index(fields=['project', 'created_at'],
name='animation_p_project_59979d_idx'),
            models.Index(fields=['project', 'frame', 'created_at'],
name='animation_p_project_6b4b56_idx'),
            models.Index(fields=['project', 'is_resolved'],
name='animation_p_project_30b060_idx'),
        ]

    def __str__(self):
        return f'{self.project} comment by {self.author}'


class Frame(models.Model):
    project = models.ForeignKey(AnimationProject, on_delete=models.CASCADE,
related_name='frames')
    index = models.PositiveIntegerField(verbose_name='Frame number')
    content_json = models.TextField(blank=True, verbose_name='Frame content JSON')
    content_revision = models.PositiveIntegerField(default=0, verbose_name='Frame
content revision')
    preview_image = models.ImageField(upload_to='frames/', blank=True, null=True,
verbose_name='Frame preview')
    created_at = models.DateTimeField(auto_now_add=True, verbose_name='Created at')
    updated_at = models.DateTimeField(auto_now=True, verbose_name='Updated at')

    class Meta:
        ordering = ['project', 'index']
        unique_together = ('project', 'index')
```

```
    def __str__(self):
        return f'{self.project.title} - frame {self.index}'


class ProjectPresenceSession(models.Model):
    project = models.ForeignKey(
        AnimationProject,
        on_delete=models.CASCADE,
        related_name='presence_sessions',
    )
    user = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        on_delete=models.CASCADE,
        related_name='project_presence_sessions',
    )
    channel_name = models.CharField(max_length=255, unique=True)
    current_frame = models.ForeignKey(
        Frame,
        null=True,
        blank=True,
        on_delete=models.SET_NULL,
        related_name='presence_sessions',
    )
    role = models.CharField(
        max_length=16,
        choices=ProjectMember.Role.choices,
        default=ProjectMember.Role.VIEWER,
    )
    joined_at = models.DateTimeField(auto_now_add=True)
    last_seen_at = models.DateTimeField(default=timezone.now)
    is_active = models.BooleanField(default=True)

    class Meta:
        indexes = [
            models.Index(fields=['project', 'is_active', 'last_seen_at']),
            models.Index(fields=['project', 'user', 'is_active']),
        ]

    def __str__(self):
        return f'{self.user} online in {self.project} ({self.channel_name})'


class FrameLock(models.Model):
    project = models.ForeignKey(
        AnimationProject,
```

```
        on_delete=models.CASCADE,
        related_name='frame_locks',
    )
    frame = models.OneToOneField(
        Frame,
        on_delete=models.CASCADE,
        related_name='frame_lock',
    )
    user = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        on_delete=models.CASCADE,
        related_name='frame_locks',
    )
    presence_session = models.ForeignKey(
        ProjectPresenceSession,
        null=True,
        blank=True,
        on_delete=models.SET_NULL,
        related_name='frame_locks',
    )
    acquired_at = models.DateTimeField(auto_now_add=True)
    last_heartbeat_at = models.DateTimeField(default=timezone.now)
    expires_at = models.DateTimeField()

    class Meta:
        indexes = [
            models.Index(fields=['project', 'expires_at']),
            models.Index(fields=['project', 'user']),
        ]

    def __str__(self):
        return f'{self.frame} locked by {self.user}'


class Layer(models.Model):
    frame = models.ForeignKey(Frame, on_delete=models.CASCADE,
related_name='layers')
    order = models.PositiveIntegerField(default=1, verbose_name='Layer order')
    name = models.CharField(max_length=200, verbose_name='Layer name')
    visible = models.BooleanField(default=True, verbose_name='Visible')
    opacity = models.PositiveSmallIntegerField(default=100, verbose_name='Opacity
(0-100)')
    content_revision = models.PositiveIntegerField(default=0, verbose_name='Layer
content revision')
```

```
    class Meta:
        ordering = ['frame', 'order', 'id']
    def __str__(self):
        return f'{self.frame} – {self.name}'


class LayerLock(models.Model):
    project = models.ForeignKey(
        AnimationProject,
        on_delete=models.CASCADE,
        related_name='layer_locks',
    )
    frame = models.ForeignKey(
        Frame,
        on_delete=models.CASCADE,
        related_name='layer_locks',
    )
    layer = models.OneToOneField(
        Layer,
        on_delete=models.CASCADE,
        related_name='layer_lock',
    )
    user = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        on_delete=models.CASCADE,
        related_name='layer_locks',
    )
    presence_session = models.ForeignKey(
        ProjectPresenceSession,
        null=True,
        blank=True,
        on_delete=models.SET_NULL,
        related_name='layer_locks',
    )
    acquired_at = models.DateTimeField(auto_now_add=True)
    last_heartbeat_at = models.DateTimeField(default=timezone.now)
    expires_at = models.DateTimeField()

    class Meta:
        indexes = [
            models.Index(fields=['project', 'frame', 'expires_at'],
name='animation_l_project_0ca8e0_idx'),
            models.Index(fields=['project', 'user'],
name='animation_l_project_542477_idx'),
```

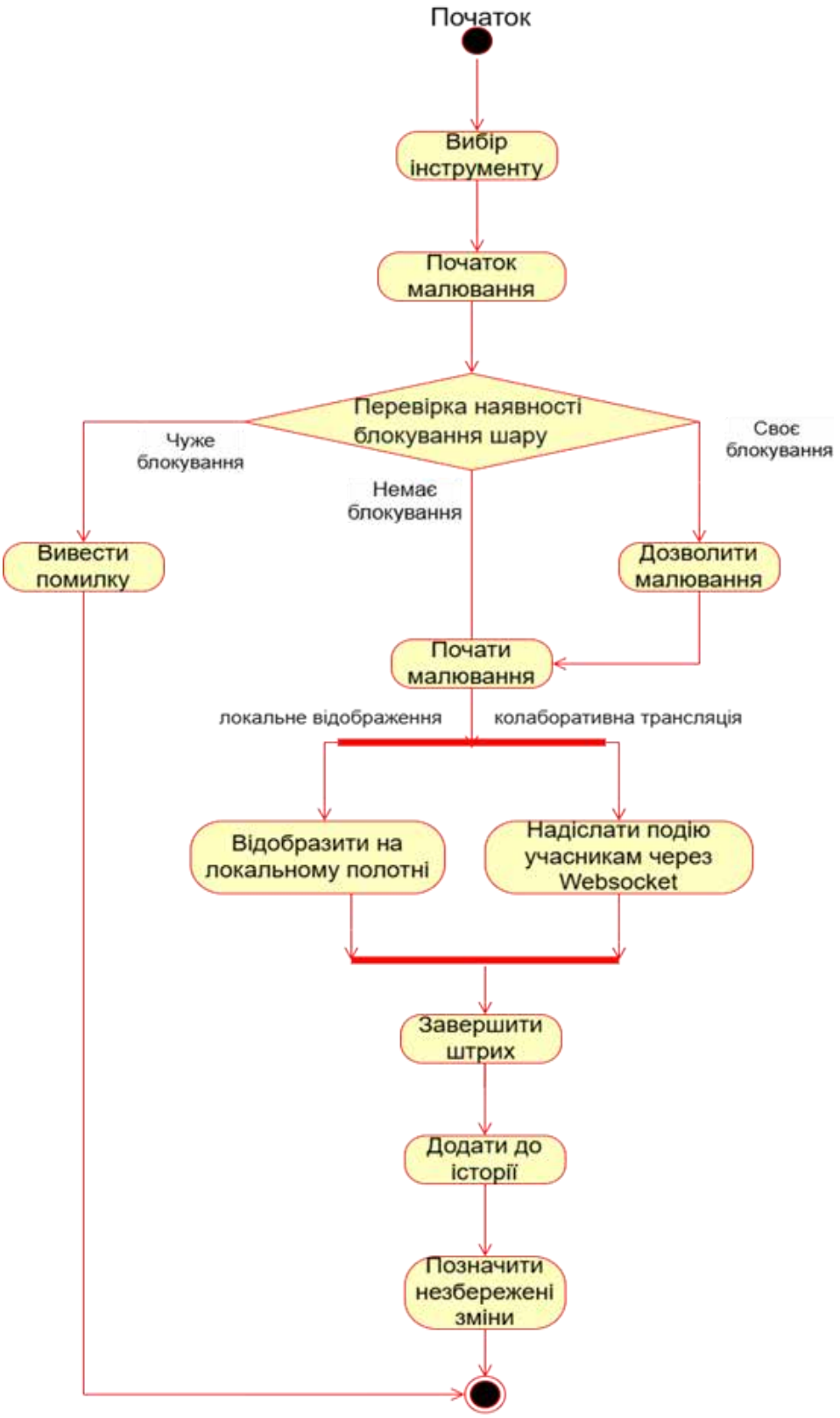
```
            models.Index(fields=['presence_session', 'expires_at'],
name='animation_l_presenc_0acc42_idx'),
        ]
    def __str__(self):
        return f'{self.layer} locked by {self.user}'
```

ДОДАТОК Б

Діаграма прецедентів програми

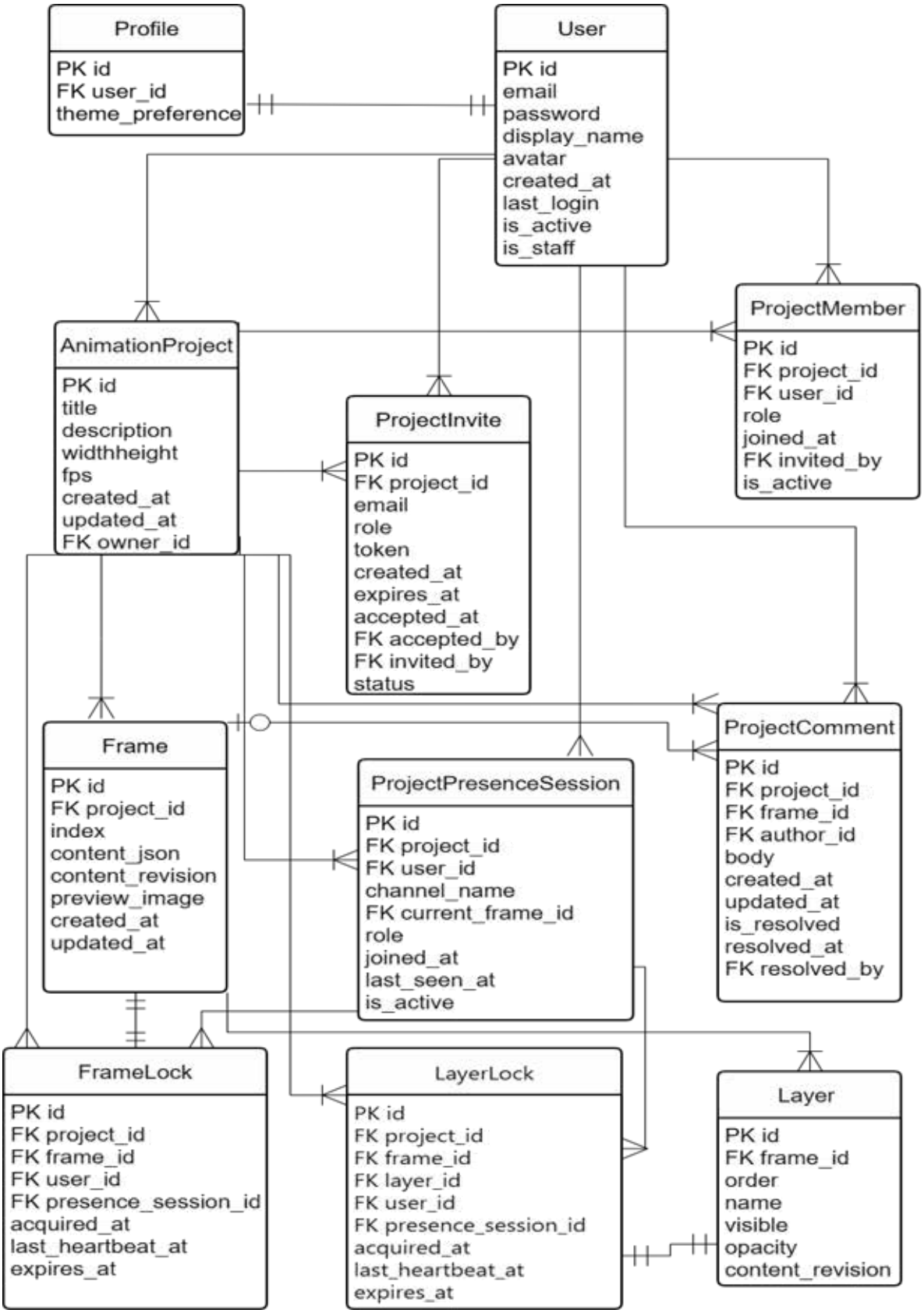
ДОДАТОК В

Діаграма діяльності для процесу «Редагування анімації в реальному часі з колаборативною підтримкою»



ДОДАТОК Д

ER діаграма логічної моделі даних



ДОДАТОК Е

Код створення та налаштування таблиць в БД

Налаштуваннями бази даних:

```
DATABASES = {
   "default": {
      "ENGINE": "django.db.backends.postgresql",
      "NAME": os.environ.get("POSTGRES_DB", "animstudio"),
      "USER": os.environ.get("POSTGRES_USER", "animstudio"),
      "PASSWORD": os.environ.get("POSTGRES_PASSWORD", ""),
      "HOST": os.environ.get("POSTGRES_HOST", "db"),
      "PORT": os.environ.get("POSTGRES_PORT", "5432"),
}
}
```

SQLite:

```
DATABASES = {
   "default": {
      "ENGINE": "django.db.backends.sqlite3",
      "NAME": BASE_DIR / "test.sqlite3",
}
}
```

Модель User:

```
class User(AbstractUser):
username = None
email = models.EmailField("Email", unique=True)
```

```
    display_name = models.CharField("Display name", max_length=150, blank=True)
    avatar = models.ImageField(upload_to="avatars/", blank=True, null=True)
    created_at = models.DateTimeField(default=timezone.now, editable=False)
    USERNAME_FIELD = "email"
    REQUIRED_FIELDS = []
    objects = UserManager()
    def __str__(self):
        return self.display_name or self.email
```

Profile:

```
class Profile(models.Model):
    class ThemePreference(models.TextChoices):
        SYSTEM = "system", "System"
        LIGHT = "light", "Light"
        DARK = "dark", "Dark"
    user = models.OneToOneField(
        settings.AUTH_USER_MODEL,
        on_delete=models.CASCADE,
        related_name="profile",
    )
      theme_preference = models.CharField(
        max_length=16,
        choices=ThemePreference.choices,
        default=ThemePreference.SYSTEM,
    )
```

AnimationProject:

```
class AnimationProject(models.Model):
```

```
        owner = models.ForeignKey(
                settings.AUTH_USER_MODEL,
                on_delete=models.CASCADE,
                related_name='animation_projects',
        )
        title = models.CharField(max_length=200, verbose_name='Project title')
        description = models.TextField(blank=True, verbose_name='Description')
        width = models.PositiveIntegerField(default=1280, verbose_name='Canvas
width (px)')
        height = models.PositiveIntegerField(default=720, verbose_name='Canvas
height (px)')
        fps = models.PositiveIntegerField(default=12, verbose_name='Frames per
second')
        created_at = models.DateTimeField(auto_now_add=True,
verbose_name='Created at')
        updated_at = models.DateTimeField(auto_now=True, verbose_name='Updated
at')


        ProjectMember:
        class ProjectMember(models.Model):
        class Role(models.TextChoices):
                OWNER = 'owner', 'Owner'
                EDITOR = 'editor', 'Editor'
                VIEWER = 'viewer', 'Viewer'
        project = models.ForeignKey(AnimationProject,
on_delete=models.CASCADE, related_name='memberships')
        user = models.ForeignKey(settings.AUTH_USER_MODEL,
on_delete=models.CASCADE, related_name='project_memberships')
        role = models.CharField(max_length=16, choices=Role.choices,
default=Role.VIEWER)
```

```
        joined_at = models.DateTimeField(auto_now_add=True)
        invited_by = models.ForeignKey(settings.AUTH_USER_MODEL, null=True,
blank=True, on_delete=models.SET_NULL,
related_name='sent_project_memberships')
        is_active = models.BooleanField(default=True)
        class Meta:
                constraints = [
                 models.UniqueConstraint(fields=['project', 'user'],
name='unique_project_member'),
                ]


        Frame:
        class Frame(models.Model):
        project = models.ForeignKey(AnimationProject,
on_delete=models.CASCADE, related_name='frames')
        index = models.PositiveIntegerField(verbose_name='Frame number')
        content_json = models.TextField(blank=True, verbose_name='Frame content
JSON')
        content_revision = models.PositiveIntegerField(default=0,
verbose_name='Frame content revision')
        preview_image = models.ImageField(upload_to='frames/', blank=True,
null=True, verbose_name='Frame preview')
        created_at = models.DateTimeField(auto_now_add=True,
verbose_name='Created at')
        updated_at = models.DateTimeField(auto_now=True, verbose_name='Updated
at')
        class Meta:
                ordering = ['project', 'index']
                unique_together = ('project', 'index')
```

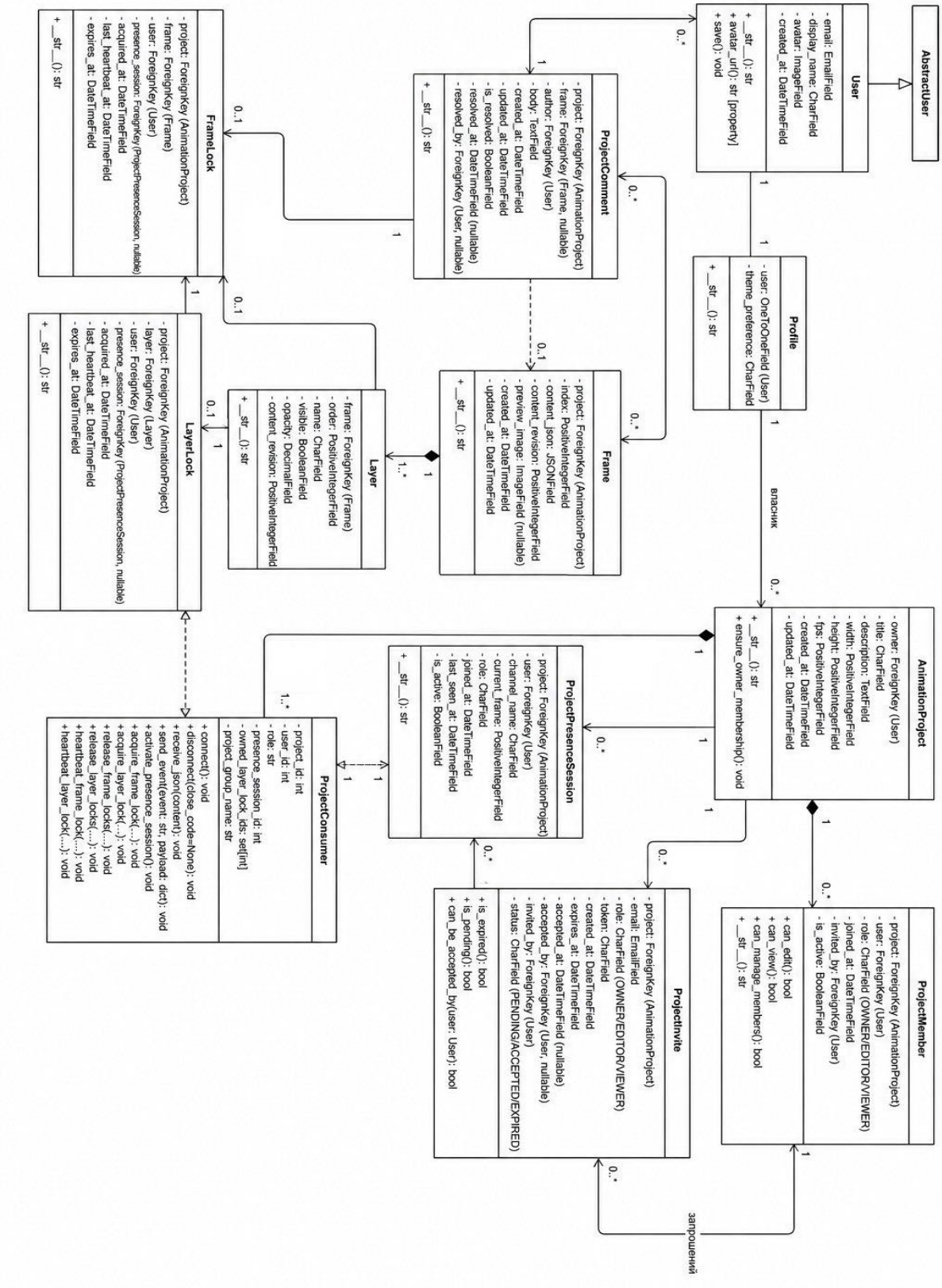
```
Layer:
class Layer(models.Model):
frame = models.ForeignKey(Frame, on_delete=models.CASCADE,
related_name='layers')
order = models.PositiveIntegerField(default=1, verbose_name='Layer order')
name = models.CharField(max_length=200, verbose_name='Layer name')
visible = models.BooleanField(default=True, verbose_name='Visible')
opacity = models.PositiveSmallIntegerField(default=100,
verbose_name='Opacity (0-100)')
content_revision = models.PositiveIntegerField(default=0,
verbose_name='Layer content revision')
class Meta:
    ordering = ['frame', 'order', 'id']

ProjectInvite:
def generate_project_invite_token():
return uuid.uuid4().hex
def default_project_invite_expiry():
return timezone.now() + timedelta(days=7)
class ProjectInvite(models.Model):
class Role(models.TextChoices):
    EDITOR = 'editor', 'Editor'
    VIEWER = 'viewer', 'Viewer'
class Status(models.TextChoices):
    PENDING = 'pending', 'Pending'
    ACCEPTED = 'accepted', 'Accepted'
    REVOKED = 'revoked', 'Revoked'
    EXPIRED = 'expired', 'Expired'
project = models.ForeignKey(AnimationProject,
on_delete=models.CASCADE, related_name='invites')
```

```
        email = models.EmailField()
        role = models.CharField(max_length=16, choices=Role.choices,
default=Role.VIEWER)
        token = models.CharField(max_length=255, unique=True,
default=generate_project_invite_token, editable=False)
        created_at = models.DateTimeField(auto_now_add=True)
        expires_at = models.DateTimeField(default=default_project_invite_expiry)
        accepted_at = models.DateTimeField(null=True, blank=True)
        accepted_by = models.ForeignKey(settings.AUTH_USER_MODEL,
null=True, blank=True, on_delete=models.SET_NULL,
related_name='accepted_project_invites')
        invited_by = models.ForeignKey(settings.AUTH_USER_MODEL,
on_delete=models.CASCADE, related_name='sent_project_invites')
        status = models.CharField(max_length=16, choices=Status.choices,
default=Status.PENDING)
```

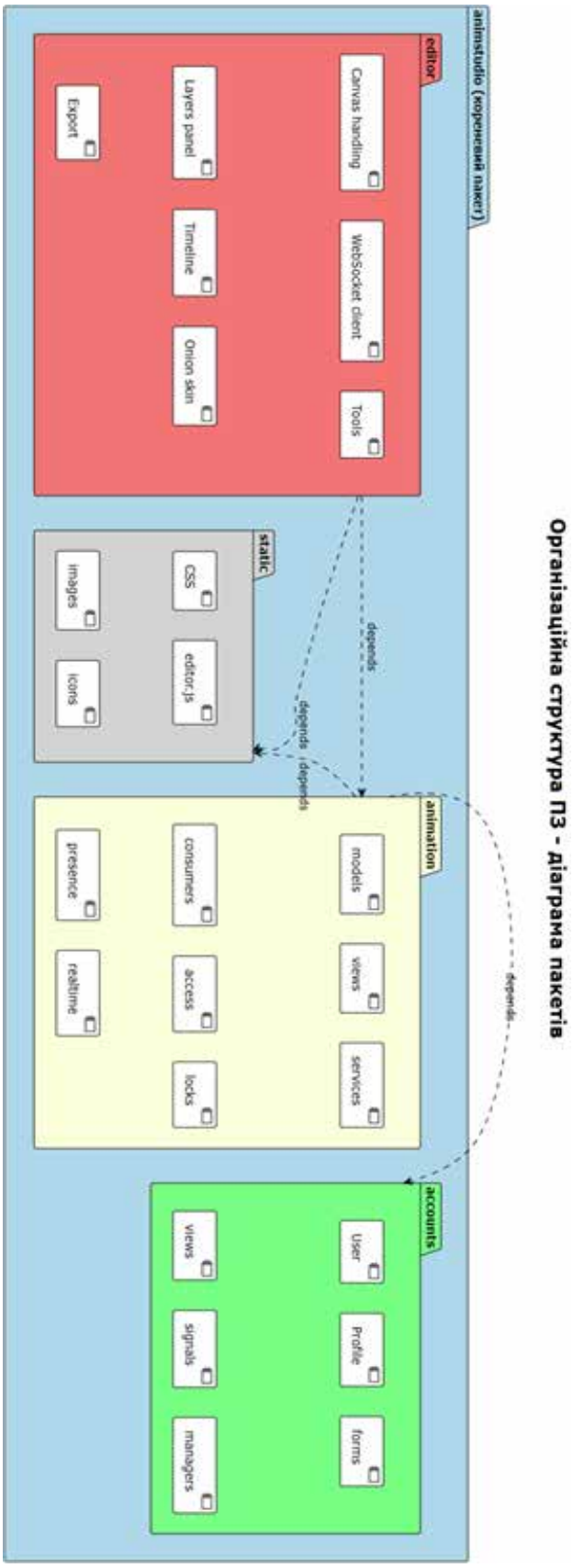
ДОДАТОК Ж

Діаграма класів програми



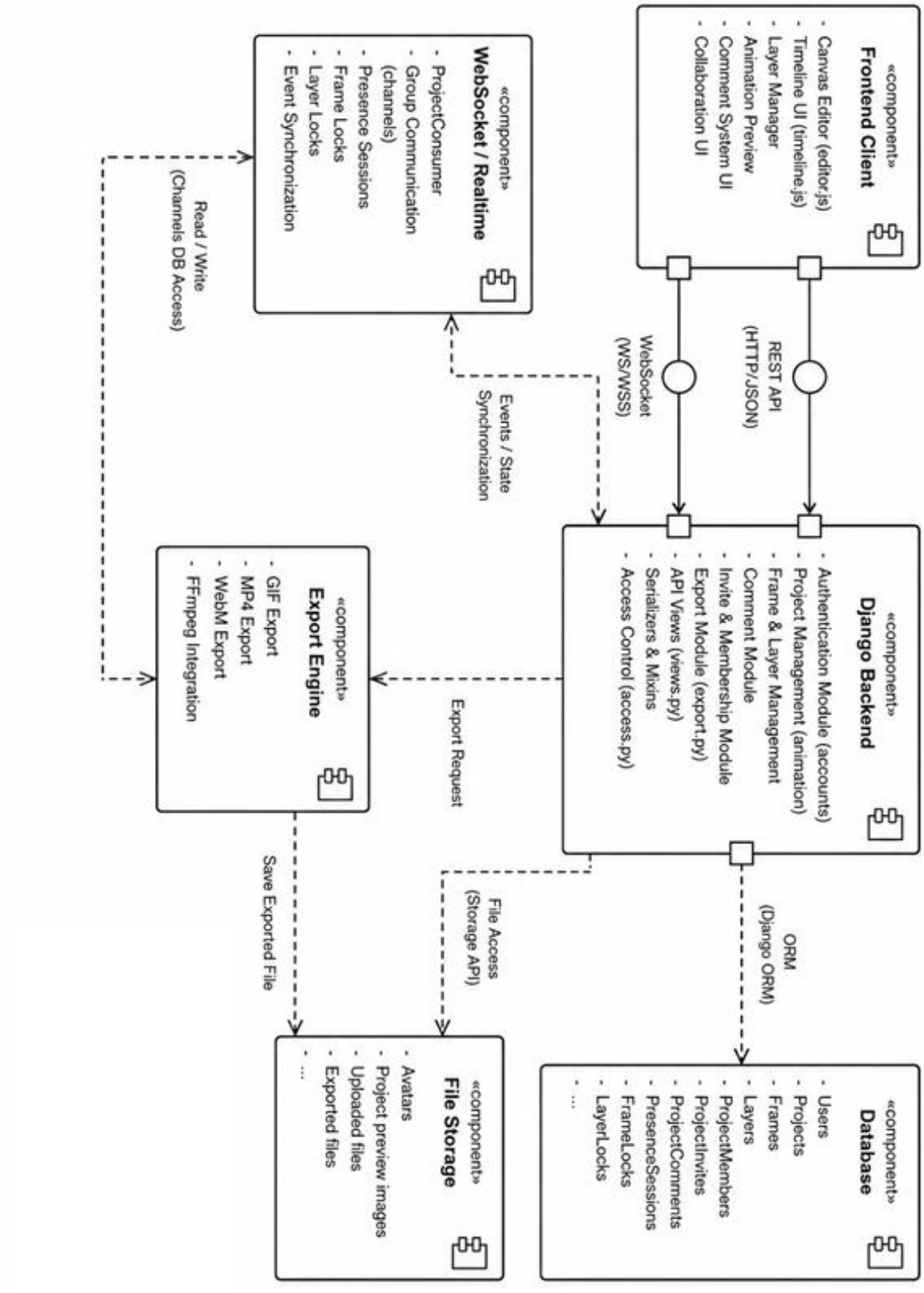
ДОДАТОК И

Діаграма пакетів програми



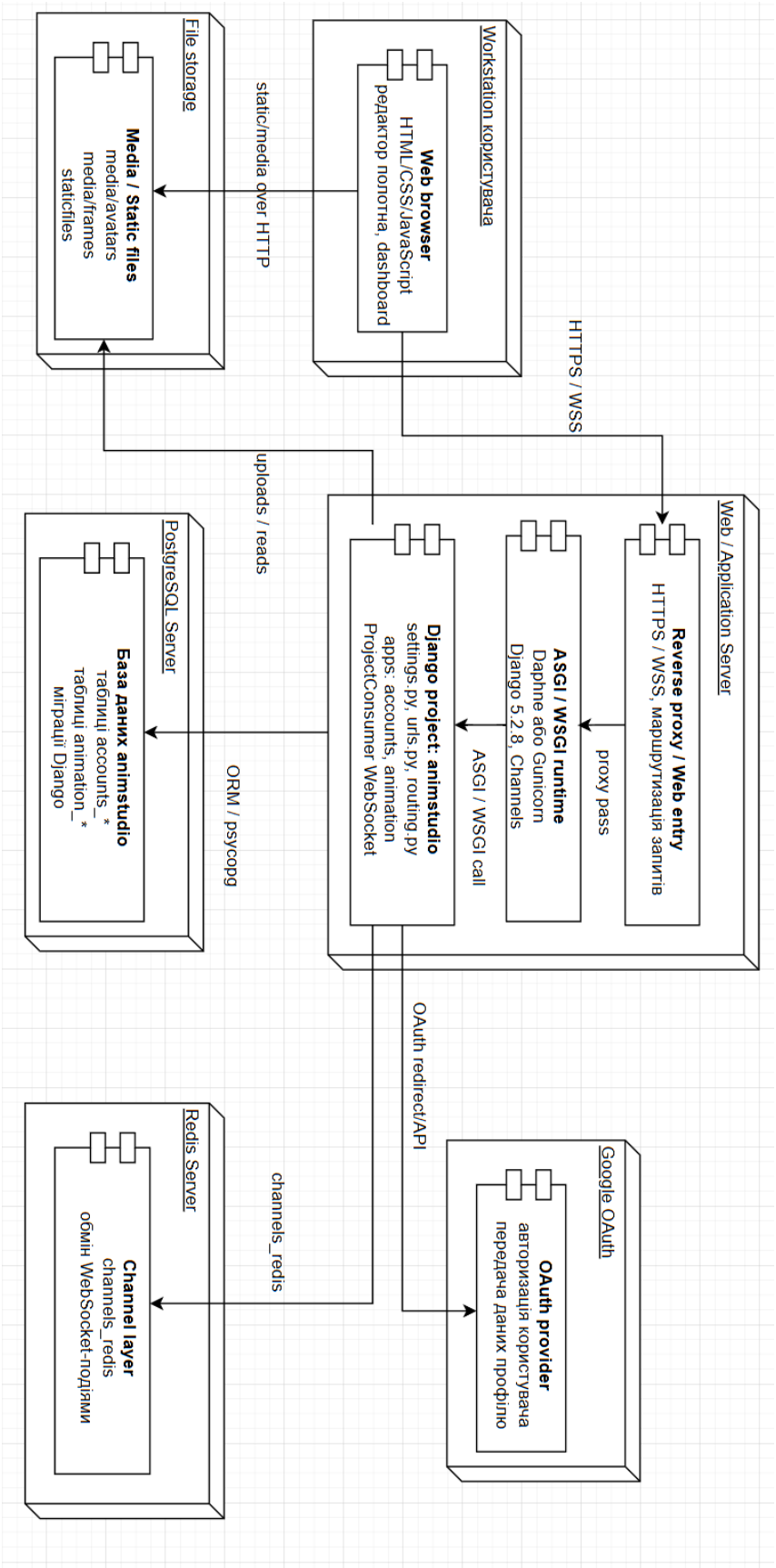
ДОДАТОК К

Діаграма компонентів



ДОДАТОК Л

Діаграма розгортання системи



ДОДАТОК М

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

Я, Герасименко Вадим Миколайович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Програмне забезпечення редактора двовимірної графіки та анімації» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):

- [ ] інструменти генеративного ШІ не використовувалися.
- [✓] інструменти ШІ (ChatGPT, Gemini) були використані для:
  - [ ] перекладу іншомовних джерел;
  - [✓] стилістичного редагування та перевірки граматики;
  - [✓] допомоги у написанні/відлагодженні програмного коду;
  - [ ] інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«___»_____ 2026 р. _____ Вадим ГЕРАСИМЕНКО