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: «Веборієнтоване програмне забезпечення для створення рекламного продукту»

Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент _____ **Ганна ВАЙГАНГ**
(підпис)

Керівник бакалаврської кваліфікаційної роботи _____ **Іван КОРНІЛОВ**
(підпис)

Консультант бакалаврської кваліфікаційної роботи
к.т.н., доцент _____ **Ганна ВАЙГАНГ**
(підпис)

Виконав _____ **Олександр ЛЕОНТОВИЧ**
(підпис)

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

Леонтовичу Олександру Сергійовичу
(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи
«Веборієнтоване програмне забезпечення для створення рекламного продукту»
затверджена наказом від «19» грудня 2025 р. № 3204 «С»

Термін подання завершеної роботи на кафедру «_22__» травня 2026 р.

Вихідні дані до роботи: матеріали проєктного практикуму, методичні рекомендації щодо підготовки та оформлення бакалаврської кваліфікаційної роботи, результати моделювання предметної області.

Перелік питань, що підлягають дослідженню: Аналіз предметної області створення рекламного продукту. Проєктування інформаційного забезпечення веборієнтованої системи. Моделювання структури, компонентів, ролей користувачів та алгоритмів роботи програмного забезпечення. Розробка веборієнтованого програмного забезпечення для створення рекламного продукту.

Перелік графічних документів (за необхідності).

Дата видачі завдання «__19__» ___грудня_____ 2025 р.

Керівник бакалаврської кваліфікаційної роботи _____ **Іван КОРНІЛОВ**
(підпис)

Консультант бакалаврської кваліфікаційної роботи _____ **Ганна ВАЙГАНГ**
(підпис)

Завдання взяв до виконання _____ **Олександр ЛЕОНТОВИЧ**
(підпис)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

API – програмний інтерфейс додатку (Application Programming Interface).

БД – база даних.

CRUD – набір операцій створення, читання, оновлення та видалення даних.

CSS – каскадні таблиці стилів.

DOM – об’єктна модель документа, що використовується браузером для представлення сторінки.

ER – модель «сутність-зв’язок» для опису структури даних.

GSAP – GreenSock Animation Platform, бібліотека JavaScript для створення анімацій.

HTML – мова гіпертекстової розмітки.

JS – JavaScript, мова програмування для клієнтської логіки вебзастосунку.

JWT – JSON Web Token, маркер доступу для автентифікації користувача.

QA – контроль якості (Quality Assurance).

RLS – Row Level Security, політики безпеки на рівні рядків у PostgreSQL/Supabase.

RPC – віддалений виклик процедури, що використовується для серверної бізнес-логіки.

SQL – мова структурованих запитів.

UI – інтерфейс користувача.

UML – уніфікована мова моделювання.

UX – користувацький досвід.

Робочий простір – середовище користувача або команди в системі Creative Workflow Studio.

ВСТУП

Сучасний ринок цифрової реклами характеризується високою швидкістю виробництва рекламних матеріалів, постійною потребою в адаптації креативів під різні формати та необхідністю координувати роботу кількох учасників виробничого процесу. Рекламний продукт уже не обмежується статичним банером: він може містити анімацію, інтерактивні елементи, HTML-структуру, JavaScript-логіку, варіанти попереднього перегляду та декілька форматів експорту. Через це створення якісного рекламного матеріалу потребує не тільки графічного редактора, а й програмного середовища, яке поєднує редагування, анімацію, перевірку, погодження та передачу результату.

Актуальність теми зумовлена тим, що у практиці креативних команд часто використовується декілька розрізнених інструментів: окремий дизайн-редактор, редактор коду, файлові сховища, таблиці для статусів, месенджери для коментарів і ручний експорт результатів. Така фрагментація ускладнює контроль актуальної версії, підвищує ризик втрати даних, робить непрозорим процес QA-перевірки та збільшує час між створенням креативу і передачею його у робоче середовище. Веборієнтоване програмне забезпечення для створення рекламного продукту дозволяє зменшити ці проблеми за рахунок єдиного середовища роботи.

У межах бакалаврської кваліфікаційної роботи розглядається програмна система Creative Workflow Studio. Вона орієнтована на повний цикл роботи з цифровими креативами:

- створення рекламного проєкту;
- редагування макета в редакторі Studio;
- додавання текстових і графічних шарів;
- налаштування анімації через безкодового редактора часової шкали або через JavaScript/GSAP;
- попередній перегляд;
- передавання на QA-перевірку;

- повернення з коментарем або погодження;
- експорт результату у відповідному форматі.

Мета роботи - спроєктувати, обґрунтувати та описати реалізацію веборієнтованого програмного забезпечення для створення рекламного продукту, яке забезпечує редагування цифрових креативів, командний робочий процес, контроль доступів, QA-перевірку, оновлення в реальному часі та експорт готового результату.

Для досягнення мети необхідно виконати такі завдання:

1. проаналізувати предметну область створення рекламного продукту та визначити ключові проблеми, які виникають у виробничого робочого процесу;
2. сформувати функціональні й нефункціональні вимоги до веборієнтованої системи з позиції користувача, власника та учасників команди;
3. розробити моделі предметної області, які відображають ролі, сценарії використання, структуру даних, компоненти та алгоритми роботи системи;
4. обґрунтувати вибір інформаційного забезпечення, СУБД і засобів реалізації логіки серверної частини;
5. описати архітектуру клієнтської частини, компонентну структуру, маршрути, сервіси, роботу стан редактора і механізми синхронізації в реальному часі;
6. розглянути алгоритми створення проєкту, передавання на QA, повернення з коментарем, погодження та експорту рекламного продукту;
7. визначити вимоги до тестування, апаратного і програмного забезпечення, складу інсталяційного пакету та розгортання системи.

Об’єктом розробки є процес створення рекламного продукту у вебсередовищі, який охоплює проєктування макета, налаштування анімації, командну взаємодію, перевірку якості та експорт результату.

Предметом розробки є веборієнтоване програмне забезпечення Creative Workflow Studio, його інформаційна структура, архітектура клієнтської частини, механізми взаємодії з Supabase, логіка робочого процесу та алгоритми керування рекламними проєктами.

Методи роботи включають аналіз предметної області, порівняння існуючих інструментів, об’єктно-орієнтоване та структурне моделювання, побудову UML і ER-діаграм, проєктування інформаційної бази, програмну реалізацію клієнтської логіки, використання хмарних сервісів модель «серверна частина як сервіс» (BaaS), функціональне тестування та узагальнення результатів.

Під час реалізації застосовано React 18 для побудови компонентного інтерфейсу, Vite для швидкої розробки та підсумкової збірки, Supabase для автентифікації, PostgreSQL, RLS, RPC, Edge Functions і Realtime, GSAP для анімаційної логіки, Monaco Editor для редагування JavaScript/CSS, а також допоміжні бібліотеки для візуальних елементів і статистики. Вибір цих засобів відповідає вимогам складного інтерактивного вебредактора та підтримує подальше розширення системи.

Практичне значення роботи полягає у створенні цілісної програмної платформи, яку можна використовувати як навчальний та демонстраційний приклад організації виробничого процесу для цифрових креативів. Рішення може бути основою для подальшого розвитку: розширення набору шаблонів, додавання історії версій, журналу аудиту, наскрізних тестів, більш гнучких ролей, покращення експорту та інтеграції з рекламними платформами.

Пояснювальна записка складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і додатків. У першому розділі розглянуто предметну область і вимоги. У другому розділі подано проєктування інформаційного забезпечення. У третьому розділі описано архітектуру та програмну реалізацію. У четвертому розділі наведено рекомендації щодо тестування, розгортання та експлуатації системи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Опис предметної області створення рекламного продукту

Предметна область цієї роботи пов’язана зі створенням рекламного продукту за допомогою веборієнтованого програмного забезпечення. У сучасному цифровому середовищі рекламний продукт може бути банером, анімованим HTML-креативом, коротким відео, GIF-презентацією, інтерактивним блоком або набором адаптованих форматів для різних рекламних майданчиків. Його підготовка включає не тільки візуальне оформлення, а й технічну реалізацію, оптимізацію ваги файлів, перевірку сценарію анімації, відповідність вимогам платформи та контроль версій [9; 14].

Класичний виробничий процес цифрової реклами зазвичай має кілька ролей: замовник або маркетолог формулює вимогу, дизайнер готує макет, розробник реалізує анімацію та інтерактивність, QA перевіряє відповідність вимогам, а керівник або власник координує статуси, призначення і передачу результату. У невеликих командах ці ролі можуть поєднуватися однією людиною, однак логіка процесу залишається схожою: створення, редагування, перевірка, погодження та експорт [14].

Веборієнтований підхід до створення рекламного продукту має низку переваг. По-перше, користувач працює без встановлення складного десктопного програмного забезпечення. По-друге, дані зберігаються централізовано в робочому просторі, що полегшує командну взаємодію. По-третє, система може поєднувати візуальне редагування, кодовий режим, анімацію за часовою шкалою, попередній перегляд і процес експорту в одному інтерфейсі. По-четверте, доступи, ролі та статуси робочого процесу можна контролювати на рівні програмної логіки і бази даних [9; 14].

У цьому підрозділі предметну область розглянуто на основі вихідних даних, функціональних вимог і моделей, підготовлених для програмної системи

Creative Workflow Studio. Основну увагу приділено тому, які об’єкти, ролі та процеси утворюють середовище створення рекламного продукту.

Предметна область даної роботи – створення рекламного продукту за допомогою веборієнтованого програмного забезпечення. Така система призначена для користувачів, які працюють із цифровою рекламою: створюють рекламні макети, додають текстові та графічні елементи, налаштовують анімацію, переглядають результат, передають рекламний продукт на перевірку та експортують готовий файл [14; 15].

Подібне програмне забезпечення може використовуватися креативними командами, маркетинговими відділами, дизайнерами, розробниками рекламних банерів та спеціалістами, які відповідають за перевірку якості рекламного матеріалу. Воно дозволяє організувати роботу над рекламним продуктом у єдиному середовищі, без постійного переходу між різними сервісами [14].

Основна ідея системи полягає в тому, щоб об’єднати кілька етапів роботи над рекламним продуктом: створення проєкту, редагування макета, додавання зображень і тексту, налаштування анімації, попередній перегляд, перевірку, погодження та експорт готового результату [14].

У предметній області важливими є такі об’єкти:

- користувач;
- робочий простір;
- рекламний проєкт;
- рекламний продукт;
- макет;
- текстовий елемент;
- графічний елемент;
- анімація;
- перевірка;
- коментар перевіряючого;
- експортований файл.

Метою розробки є створення веборієнтованої системи, яка дозволяє користувачам створювати рекламні продукти в онлайн-середовищі. Система має підтримувати повний цикл роботи з рекламним проєктом: від формування макета до перевірки, погодження та експорту готового результату [14].

Creative Workflow Studio вирішує задачу об’єднання редактора креативів і виробничого робочого процесу. Система не є лише графічним редактором; вона поєднує логіку робочого простору, ролі, статуси, коментар QA, повідомлення в реальному часі та механізми експорту. Це дозволяє розглядати її як програмний продукт, що автоматизує важливу частину життєвого циклу рекламного матеріалу [14].

1.2. Аналіз вимог до програмної системи

Вимоги до системи доцільно розглядати з кількох позицій: адміністратора або власника робочого простору, кінцевого користувача, розробника, QA-фахівця та керівника команди. Під власником робочого простору в роботі розуміється користувач, який створює командний простір, керує його учасниками, ролями, доступами та загальним станом проєктів [14; 15].

Власник робочого простору Creative Workflow Studio має отримати стабільне веборієнтоване програмне забезпечення для створення рекламного продукту, у якому об’єднано редактор креативів, командний робочий процес, QA-перевірку, попередній перегляд, експорт та керування робочим простором. Система повинна підтримувати повний цикл роботи: створення рекламного проєкту, редагування макета, додавання текстових і графічних шарів, налаштування анімації, передавання на перевірку, погодження або повернення на доопрацювання та експорт готового результату [14; 15].

З точки зору власника важливо, щоб клієнтська частина на React 18 і Vite працювала швидко, коректно збиралася в підсумкову збірку та могла розміщуватися на статичному хостингу. Логіка серверної частини реалізується через Supabase: Auth, PostgreSQL, Row Level Security, Realtime, RPC та Edge

Functions. Такий підхід дозволяє не підтримувати окремий класичний сервер, але водночас мати авторизацію, ролі, таблиці, оновлення в реальному часі та серверні операції для складніших сценаріїв [3; 4; 5; 6; 8; 14].

До програмних вимог власника належать:

- встановлені Node.js LTS і npm;
- репозиторій проєкту з Vite/React-застосунком;
- Supabase-проєкт із налаштованими таблицями profiles, workspaces, workspace_members, projects, project_notifications, workspace_payments;
- виконані SQL-скрипти для RLS-політик;
- доступ до панелі керування Supabase;
- файл .env з потрібними ключами доступу до клієнтського середовища [3; 4; 5; 10; 14].

Апаратні вимоги для розробки та демонстрації: персональний комп’ютер або ноутбук з 8 ГБ ОЗП, сучасним процесором, 10 ГБ вільного місця на диску та стабільним підключенням до Інтернету. Для тестування бажано мати сучасний браузер, Git, VS Code або інше середовище розробки, а також доступ до Supabase і GitHub Pages або іншого хостингу для демонстраційного розгортання [14; 15].

Також власник системи очікує надійного контролю доступів. Власник і Керівник мають керувати робочим простором, учасниками та призначеннями, Розробник має редагувати призначені йому проєкти, а QA повинен мати доступ до попереднього перегляду, перевірки, погодження або повернення креативу з коментарем. Дані проєктів не повинні бути доступні стороннім користувачам, тому перевірки ролей реалізуються на рівні клієнтської частини для зручності користувача та на рівні Supabase RLS/RPC для безпеки [14; 15].

Користувачеві потрібен пристрій із сучасним браузером, доступом до Інтернету, дисплеєм достатнього розміру, клавіатурою та мишею або тачпадом. Для комфортної роботи зі редактором Studio бажано використовувати ноутбук або ПК, оскільки редактор містить робоче полотно, панель властивостей, вікно шарів, панелі JS/CSS/часова шкала та попередній перегляд [14; 15].

З програмної точки зору користувач має відкрити застосунок у браузері, створити обліковий запис або увійти в існуючий, пройти первинне налаштування робочого простору або приєднатися до командного робочого простору за запрошенням. Після цього користувач отримує доступ до панелі проєктів зі списком рекламних проєктів, фільтрами, статусами, призначеного розробникааа/QA та діями попередній перегляд, експорт, Open, Clone і Delete [14; 15].

Для розробника основною вимогою є можливість швидко створити або відкрити рекламний проєкт, додати зображення й текстові шари, змінити позицію, розміри, прозорість, обертання, фон сцени, написати або відредагувати CSS/JavaScript, налаштувати GSAP-анімацію через безкодового редактора часової шкали і передати готовий креатив на QA-перевірку [14; 15].

Для QA-користувача важливо мати можливість відкрити попередній перегляд, перевірити відповідність рекламного продукту вимогам, переглянути анімацію, формат і експорт, після чого або перевести проєкт у стан готовності до використання, або повернути його у стан розроблення із конкретним коментарем. Для Власника і Керівника важливо бачити загальний стан роботи команди, призначати відповідальних осіб, контролювати статуси та отримувати повідомлення про зміни в проєктах [14; 15].

У межах аналізу вимог розглянуто основні алгоритми, які відповідають логіці веборієнтованого програмного забезпечення для створення рекламного продукту:

- створення нового проєкту;
- передавання креативу на QA-перевірку;
- експорт готового рекламного продукту.

Загальні функціональні вимоги до системи подано у табл. 1.1. Вони згруповані за основними сценаріями: робота з обліковим записом, керування робочим простором, створення і редагування проєкту, QA-перевірка, експорт, повідомлення та контроль доступів.

Таблиця 1.1

Функціональні вимоги до програмної системи

Група вимог	Зміст вимоги	Очікуваний результат
Авторизація	Реєстрація, вхід, підтвердження пошти, гостьовий режим	Користувач отримує відповідний рівень доступу до системи
Робочий простір	Створення персонального/командного робочого простору, приєднання за запрошенням, керування ролями	Проєкти й доступи ізольовані в межах робочого простору
Проєкти	Створення, клонування, відкриття, видалення та фільтрація рекламних проєктів	Команда бачить актуальний список креативів і статусів
Редактор	Робоче полотно, шари, панеллю властивостей, JS/CSS редактором, редактор часової шкали	Користувач редагує макет і анімацію в єдиному середовищі
QA-перевірка	Передавання на QA-перевірку, підтвердження, повернення з коментарем	Процес перевірки стає прозорим і контрольованим
Експорт	HTML, GIF, MP4, попередній перегляд	Результат можна передати або продемонструвати в потрібному форматі
Безпека	RLS, RPC, перевірка доступів та ролей, перевірка участі у робочому просторі	Дані робочого простору не доступні стороннім користувачам

Нефункціональні вимоги до системи подано у табл. 1.2. Вони визначають якість роботи інтерфейсу, безпечність доступу, стабільність збереження даних і можливість розгортання системи на віддаленому хостингу [14; 15].

Таблиця 1.2

Нефункціональні вимоги до програмної системи

Група вимог	Зміст вимоги	Очікуваний результат
Продуктивність	Швидке відкриття основних сторінок, стабільна робота редактора та таблиці проєктів	Користувач може працювати з редактором без помітних затримок у типових сценаріях
Зручність використання	Зрозумілі маршрути, кнопки дій, статуси проєкту, повідомлення та модальні вікна	Користувач виконує базові дії без додаткового навчання
Адаптивність	Коректне відображення панелі проєктів, кабінету, попереднього перегляду та службових вікон	Інтерфейс зберігає читабельність на різних розмірах екрана
Безпека	RLS-політики, перевірка членства в робочому просторі, обмеження дій за ролями	Дані робочого простору не доступні стороннім користувачам
Надійність збереження	Збереження стану редактора, статусів і повідомлень у базі даних	Проєкт відновлюється після перезавантаження сторінки
Розгортання	Можливість сформувати підсумкову збірку та розмістити її на статичному хостингу	Систему можна демонструвати через віддалене HTTPS-посилання

1.3. Моделювання предметної області

Моделювання предметної області дозволяє формалізувати сценарії, ролі та структуру системи до початку детальної реалізації. Для опису системи використано комплекс UML-моделей: діаграму прецедентів, діаграму послідовності, діаграму активності, діаграму абстракцій, діаграми класів, пакетів, компонентів і розгортання.

Основні функції системи та взаємодію користувача з ними подано на рис. 1.1.

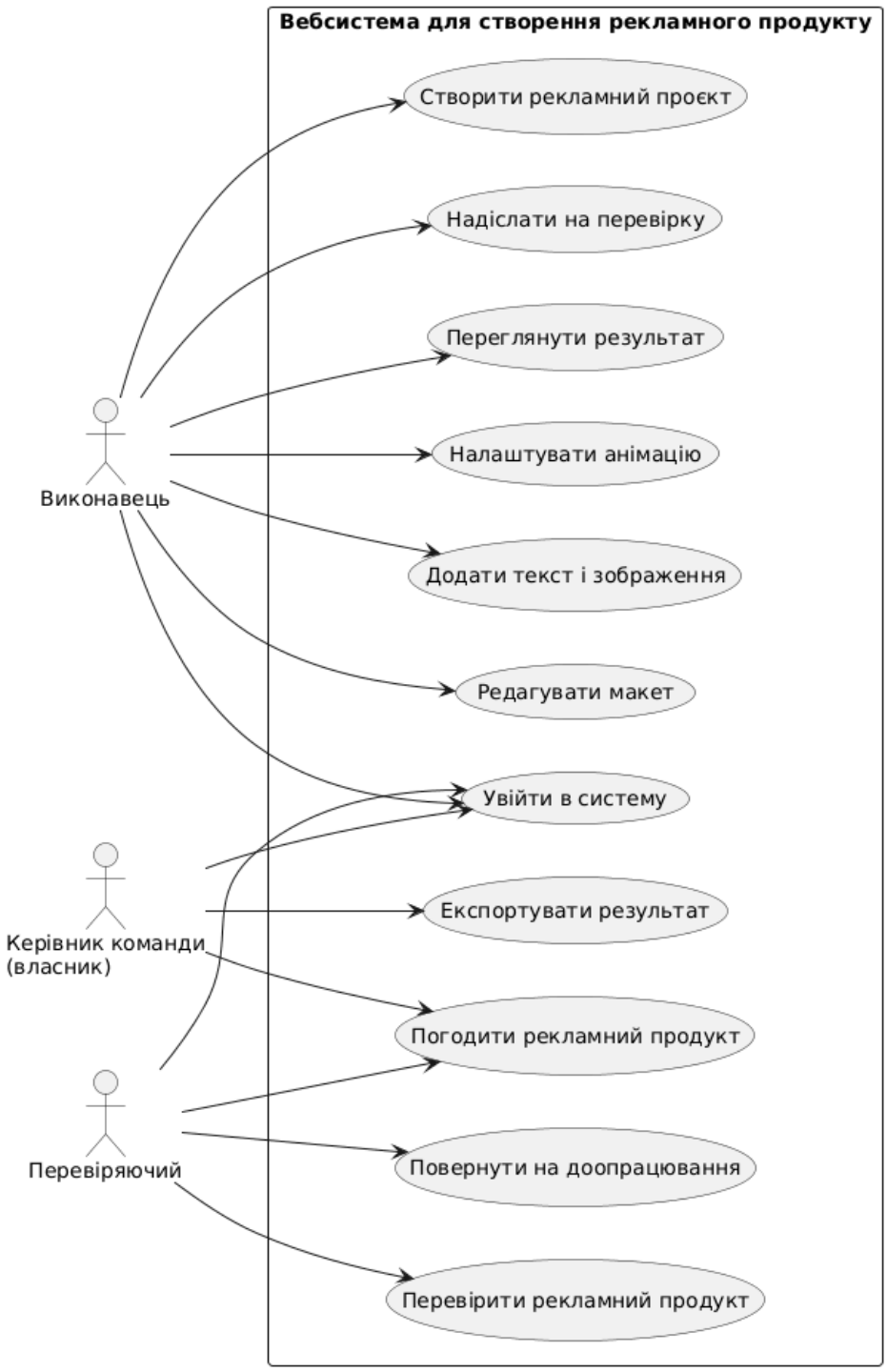

Рис. 1.1 Діаграма прецедентів веборієнтованої системи створення рекламного продукту

Діаграма прецедентів відображає основні можливості веборієнтованої системи для створення рекламного продукту та показує, які дії може виконувати користувач у межах програмного забезпечення. На рис. 1.1 узагальнено подано

взаємодію користувача із системою: реєстрацію та вхід, вибір або створення робочого простору, створення нового рекламного проєкту, відкриття редактора, редагування макета, додавання текстових і графічних елементів, керування шарами, налаштування анімації, попередній перегляд результату, передавання проєкту на перевірку, отримання коментаря після перевірки, погодження рекламного продукту та експорт готового результату.

Центральним актором на діаграмі виступає користувач системи, оскільки саме він ініціює основні сценарії роботи з рекламним продуктом. Окремо на діаграмі показано виконавця, перевіряючого та керівника команди; роль власника робочого простору у цій моделі відображена через адміністративні дії керівника команди, який координує погодження, експорт і контроль доступу. Такий підхід дозволяє показати систему на загальному рівні без перевантаження діаграми великою кількістю ролей. У межах цього узагальненого актора можуть діяти різні типи користувачів: адміністратор робочого простору, розробник, QA-фахівець і керівник команди. Кожна з цих ролей має власний рівень доступу, однак усі вони взаємодіють із тими самими базовими функціональними блоками системи: робочим простором, рекламними проєктами, редактором, перевіркою та експортом.

Окрему групу прецедентів становлять дії, пов’язані з підготовкою рекламного проєкту. Після створення проєкту користувач може перейти до редагування макета, де виконується додавання елементів, зміна їхніх параметрів, налаштування розташування, розмірів, стилів і порядку шарів. Ці дії формують основу рекламного продукту, оскільки саме в редакторі визначається візуальна структура майбутнього креативу. Додатково система підтримує налаштування анімації, що є важливою функцією для digital-реклами, оскільки рекламний продукт часто має бути не лише статичним, а й динамічним.

Ще одна важлива група прецедентів пов’язана з перевіркою якості та погодженням результату. Після завершення редагування рекламний проєкт може бути переданий на QA-перевірку. На цьому етапі перевіряється відповідність рекламного продукту вимогам, коректність відображення макета, робота

анімації та можливість подальшого експорту. Якщо у проєкті виявлено недоліки, він повертається на доопрацювання з коментарем. Якщо результат відповідає вимогам, проєкт може бути погоджений і переведений у стан готовності до експорту. Такий сценарій демонструє, що система підтримує не лише редагування, а й повний робочий процес створення рекламного продукту.

Завершальним прецедентом є експорт готового результату. Він передбачає перетворення створеного в системі рекламного проєкту у формат, придатний для подальшого використання, демонстрації або передачі. Таким чином, діаграма прецедентів показує повний шлях роботи користувача із системою: від входу та створення робочого середовища до формування, перевірки, погодження й експорту рекламного продукту. Розподіл прав між окремими ролями детальніше розглядається у подальших підрозділах, де описано функціональні вимоги та рольову модель системи.

Послідовність створення та редагування рекламного проєкту подано на рис. 1.2. Перевірку й доопрацювання рекламного продукту наведено на рис. 1.3, а погодження та експорт готового результату – на рис. 1.4.

Рис. 1.2 Діаграма послідовності створення та редагування рекламного проєкту

Рис. 1.3 Діаграма послідовності перевірки та доопрацювання рекламного продукту

Рис. 1.4 Діаграма послідовності погодження та експорту рекламного продукту

Діаграми послідовності на рис. 1.2–1.4 уточнюють основні сценарії створення, перевірки та експорту рекламного проєкту.

Діаграми послідовності уточнюють основний сценарій створення і перевірки рекламного проєкту. Спочатку користувач ініціює створення проєкту в інтерфейсі, після чого клієнтська частина перевіряє активний робочий простір і передає дані до Supabase. Після збереження запису проєкт відкривається в редакторі, де користувач змінює шари, стилі й анімацію. Далі проєкт переводиться у стан QA-перевірки, для відповідального QA-фахівця формується повідомлення, а після перевірки система або повертає проєкт у розробку з коментарем, або переводить його до стану готовність до використання.

Життєвий цикл рекламного проєкту у межах робочого процесу наведено на рис. 1.5.

Рис. 1.5 Діаграма активності робочого процесу рекламного проєкту

Діаграма активності уточнює життєвий цикл рекламного проєкту. Типовий сценарій починається зі створення проєкту у статусі розроблення. Після редагування розробник передає його на QA-перевірку. QA може підтвердити готовність і перевести проєкт у стан готовності до використання або повернути його у стан розроблення із коментарем. Після погодження користувач виконує експорт у потрібному форматі.

1.4. Огляд існуючих рішень та інструментів

У предметній області створення рекламного продукту існує кілька типів рішень: графічні редактори, редактори коду, системи управління задачами, платформи для попередній перегляд, хмарні сховища, інструменти анімації та спеціалізовані ad-tech сервіси. Кожен із цих інструментів закриває окрему потребу, однак не завжди забезпечує єдину модель робочого процесу від створення креативу до QA-перевірки та експорту [9; 14].

Порівняння існуючих рішень та інструментів створення рекламного продукту наведено у табл. 1.3.

Таблиця 1.3

Порівняння існуючих рішень та інструментів створення рекламного продукту

Тип рішення	Переваги	Недоліки у контексті теми
Графічні редактори	Зручне створення макета, робота з шарами, візуальні інструменти	Не забезпечують повний кодовий експорт і робочий процес і рольову QA-перевірку
Редактори коду	Гнучка реалізація HTML/CSS/JS, контроль структури	Недоступні для нетехнічних користувачів без технічних навичок
Таск-трекери	Контроль задач і відповідальних осіб	Не містять редактора креативів і попередній перегляд/експорт анімації
Хмарні BaaS-платформи	Авторизація, база даних, у реальному часі, серверні-функції	Потребують правильного проєктування політик доступу та структури даних
Creative Workflow Studio	Поєднання редактора, робочого процесу, QA, ролей, попереднього перегляду та експорту	Потребує подальшого розвитку історію версій, журналу аудиту та розширеного тестування

Порівняння показує, що мета розробки полягає не у заміні всіх професійних інструментів, а у створенні інтегрованого середовища, яке

демонструє повний цикл роботи з рекламним продуктом. Особливу цінність має поєднання безкодової анімації з можливістю ручного редагування коду, оскільки це знижує бар’єр входу для нетехнічних користувачів і водночас залишає гнучкість для розробник-користувачів [14].

1.5. Постановка завдання

Поставлена задача полягає у розробленні веборієнтованої системи Creative Workflow Studio, яка забезпечує створення, редагування, перевірку та експорт рекламного продукту в межах персонального або командного робочого простору. Система має підтримувати:

- авторизацію користувачів;
- рольову модель;
- створення проєктів;
- роботу з шарами;
- анімацію;
- попередній перегляд;
- експорт;
- коментар QA;
- повідомлення в реальному часі [14; 15].

Для реалізації поставленої задачі необхідно виконати такі етапи:

1. створити структуру застосунку з основними маршрутами: авторизацію, первинне налаштування, список проєектів, кабінет, редактор Studio і попередній перегляд;
2. реалізувати панелі проєктів зі списком проєктів, фільтрами, статусами, призначеннями та діями над проєктом;
3. спроєктувати редактор Studio із робочим полотном, шарами, панеллю властивостей, JS/CSS редактором і безкодового редактора часової шкали;

4. забезпечити статуси робочого процесу розроблення, QA-перевірка, готовність до використання, архівний стан і правила переходів між ними;
5. реалізувати рольову модель власник, керівник, розробник, QA та гість із перевірками доступів;
6. підключити Supabase Auth, PostgreSQL, RLS, RPC, Edge Functions і Realtime для зберігання даних та синхронізації;
7. передбачити експорт HTML/GIF/MP4 і імпорт HTML для повторного використання креативів;
8. описати тестування, апаратні вимоги, програмні вимоги та розгортання на віддаленому хостингу.

Бізнес-процес створення рекламного продукту у Creative Workflow Studio доцільно розглядати як послідовність взаємопов'язаних етапів, а не як одну дію редагування макета. Спочатку користувач визначає контекст роботи, тобто активний робочий простір, тип рекламного проєкту, відповідальних осіб і базові параметри креативу. Після цього формується початкова структура документа: сцена, розміри, фон, набір шарів, текстові та графічні елементи, а також службові поля, необхідні для подальшого збереження та експорту. Саме на цьому етапі закладається узгодженість між візуальним поданням рекламного продукту і його технічною структурою [14; 15].

Наступний етап пов'язаний із наповненням та редагуванням креативу. Користувач змінює положення елементів, їхні розміри, прозорість, обертання, типографіку, кольори та параметри сцени. Важливо, що в редакторі такі дії не повинні сприйматися як ізольовані зміни DOM-елементів; вони мають відображатися у внутрішній моделі документа, щоб після перезавантаження сторінки або відкриття проєкту іншим учасником система могла відновити ідентичний стан. Тому стан редактора є не допоміжним об'єктом, а центральною частиною рекламного проєкту [14; 15].

Окреме місце у процесі займає анімація. У традиційному підході анімований рекламний банер часто потребує залучення розробника, який вручну

створює GSAP- або CSS-анімацію. У запропонованій системі цей бар'єр зменшується завдяки безкодового редактора часової шкали. Користувач може вибрати елемент, шаблон анімації, тривалість, затримку та порядок виконання, а програма формує технічний опис анімації. Водночас наявність JavaScript-редактора залишає можливість розширення для користувачів, яким потрібна складніша поведінка або нестандартний сценарій руху [7; 14].

Після створення та анімації рекламний продукт переходить до стадії перевірки. На цьому етапі важливо зафіксувати не тільки факт передавання на QA-перевірку, а й контекст перевірки: хто виконав дію, кому призначено перевірку, який поточний статус проєкту, чи був збережений останній стан редактора і чи є службові коментарі. Така деталізація потрібна для прозорості командної роботи. Якщо QA повертає проєкт із зауваженнями, коментар повинен зберігатися разом зі зміною статусу, щоб розробник розумів причину повернення [14; 15].

Завершальна частина бізнес-процесу пов'язана з погодженням і експортом. Погоджений рекламний продукт не повинен випадково змінюватися користувачем, який не має відповідних прав. Тому статус готовності до використання виконує не лише інформаційну роль, а й роль обмежувача робочий процес. Експорт у HTML, GIF або MP4 є окремою операцією, що перетворює внутрішній стан редактора на файл або демонстраційний матеріал. У такому підході рекламний продукт проходить повний цикл: від ідеї та макета до технічно підготовленого результату [14].

Отже, вимоги до системи охоплюють не лише елементи інтерфейсу, а й правила зміни станів, збереження даних, обмеження доступу та інформування учасників команди. Такий підхід дозволяє розглядати Creative Workflow Studio як систему підтримки повного робочого процесу створення рекламного продукту, а не лише як окремий візуальний редактор [14; 15].

Роль власника у системі є найбільш повною, оскільки вона пов'язана з володінням робочим простором, керуванням учасниками та загальним контролем командної роботи. Власник має право створювати робочий простір,

активувати командний план, переглядати склад команди, змінювати ролі, видаляти учасників і контролювати доступ до проєктів. У реальному виробничому процесі така роль відповідає керівнику команди або власнику студії, який не обов’язково сам редагує кожний креатив, але відповідає за доступність і порядок роботи системи [14].

Роль керівника є проміжною між власником і виконавцями. Вона потрібна для команд, де керівник проєкту або технічний лід має контролювати призначення розробника і QA, переглядати статуси, координувати повернення з перевірки та стежити за дотриманням робочого процесу. Керівник не обов’язково має всі адміністративні права власника, однак повинен мати достатній рівень доступу для організації виробничого процесу. Такий поділ ролей робить систему гнучкішою і ближчою до реальної структури виробничої команди [14].

Розробник у Creative Workflow Studio відповідає за створення та технічне редагування рекламного продукту. Його головні дії пов’язані з редактором, шарами, робочим полотном, CSS/JavaScript-кодом, анімаціями за часовою шкалою та підготовкою проєкту до перевірки. Водночас розробник не повинен мати необмежений доступ до всіх робочих просторів або адміністративних дій. Обмеження прав зменшує ризик випадкового видалення проєктів, зміни чужих призначень або неконтрольованого переведення матеріалів у стан готовності до використання [14].

QA-користувач виконує роль незалежної перевірки. Для цієї ролі важливо мати доступ до попереднього перегляду, статусу робочого процесу, можливості погодження або повернення проєкту з коментарем. QA не повинен змінювати документ креативу так само вільно, як розробник, оскільки його завдання полягає у перевірці відповідності рекламного продукту вимогам. Тому система має підтримувати окремий сценарій review: перегляд, оцінювання, фіксація коментаря і зміна статусу відповідно до результату перевірки [14].

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Логічна модель даних у вигляді ER-діаграми

Інформаційне забезпечення Creative Workflow Studio має відображати не лише список рекламних проєктів, а й складну систему зв’язків між користувачами, робочими просторами, ролями, платежами, повідомленнями та даними редактора. Логічна модель даних повинна підтримувати персональні й командні робочі простори, участь у робочому просторі, призначення розробника і QA-рецензента, статуси робочого процесу, коментарі, повідомлення та дані редактора креативу [14; 15].

Логічну структуру основних об’єктів предметної області наведено на рис. 2.1.

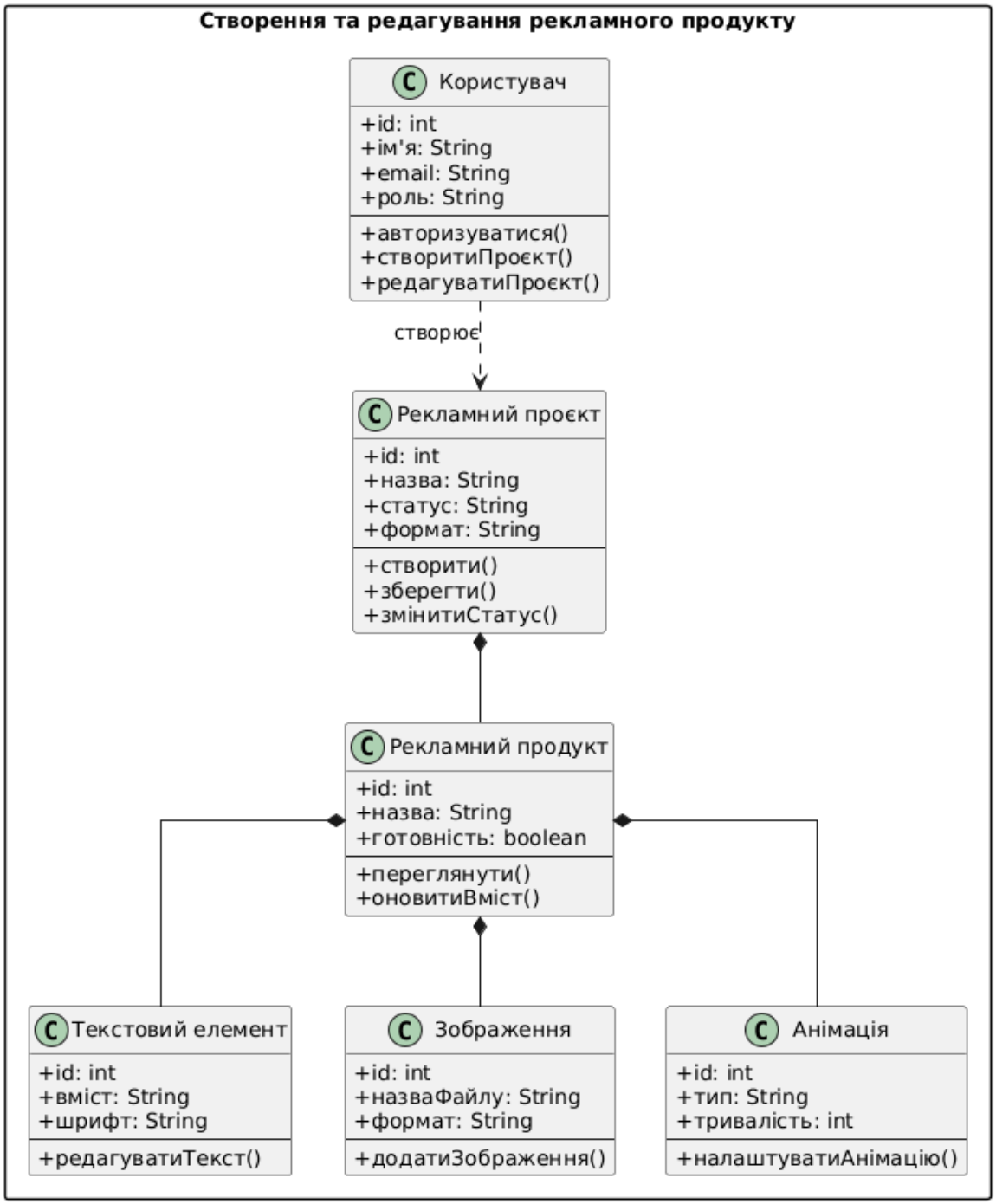

Рис. 2.1 Логічна модель основних об’єктів веборієнтованої системи

На рис. 2.1 основними об’єктами логічної моделі є Користувач, Рекламний проєкт, Рекламний продукт, Текстовий елемент, Зображення та Анімація. У фізичній структурі бази даних цим об’єктам відповідають таблиці profiles, workspaces, workspace_members, projects, project_notifications і workspace_payments, які забезпечують збереження профілів користувачів, робочих просторів, проєктів, повідомлень і платіжних даних [10; 14].

Модель даних орієнтована на реляційний підхід, оскільки між основними сутностями існують стійкі зв’язки «один-до-багатьох» і «багато-до-багатьох». Робочий простір може містити багато проєктів і багато учасників. Користувач може бути учасником декількох робочих просторів. Проєкт має одного власника, може мати призначеного розробникаа та QA-рецензента, а також багато пов’язаних повідомлень. Така структура природно реалізується в PostgreSQL через зовнішні ключі, індекси й політики доступу.

Основні сутності логічної моделі даних наведено у табл. 2.1.

Таблиця 2.1

Основні сутності логічної моделі даних

Сутність	Призначення	Ключові зв’язки
profiles	Профілі користувачів, пов’язані з авторзацією	1:N з projects через поля owner_id/developer_id/qa_id
workspaces	Персональні або командні робочі простори	1:N з projects, 1:N з workspace_members
workspace_members	Учасники робочого простору та їхні ролі	N:1 до profiles, N:1 до workspaces
projects	Рекламні проєкти, стан редактора, робочий процес	N:1 до workspaces, N:1 до profiles
project_notifications	Повідомлення щодо проєктів, QA, статусів	N:1 до projects, N:1 до profiles
workspace_payments	Платіжні сесії та активація робочого простору	N:1 до workspaces або owner profile

2.2. Вибір системи управління базами даних

Для зберігання даних обрано PostgreSQL у складі платформи Supabase. Такий вибір пояснюється поєднанням реляційної надійності, підтримки складних зв’язків, SQL-запитів, тригерів, збережених процедур, RLS-політик і зручної інтеграції з клієнтським застосунком через JavaScript-клієнт Supabase. PostgreSQL добре підходить для задач, де важливо забезпечити цілісність даних і керований доступ за ролями [4; 10; 14].

Перевагою Supabase є поєднання бази даних, автентифікації, синхронізації в реальному часі, віддалених процедур і Edge Functions у межах однієї платформи. Це зменшує потребу в підтримці окремого класичного сервера, але дає змогу виконувати серверні перевірки для критичних операцій. Відкритий клієнтський ключ не повинен надавати повний доступ до даних, тому для таблиць, доступних через API, необхідно використовувати RLS-політики [4; 5; 6; 8; 10].

Обґрунтування вибору PostgreSQL/Supabase наведено у табл. 2.2.

Таблиця 2.2

Обґрунтування вибору PostgreSQL/Supabase

Критерій	PostgreSQL/Supabase	Пояснення
Структуровані зв’язки	Так	Підтримка зовнішніх ключів, індексів, зв’язків між таблицями
Безпека на рівні даних	Так	RLS дозволяє обмежувати доступ до рядків залежно від користувача
Робота в реальному часі	Так	Можливість слухати зміни в таблицях і оновлювати інтерфейс без перезавантаження
Серверна логіка	Так	RPC та Edge Functions для складних операцій і перевірок
Інтеграція з клієнтська частина	Так	supabase-js забезпечує доступ до авторизації, бази данних і роботи в реальному часі
Масштабування прототипу	Достатнє	Архітектура підходить для навчального та демонстраційного продукту

2.3. Розробка структури інформаційної бази

Структура таблиці projects має особливе значення, оскільки вона зберігає як службову інформацію робочого процесу, так і дані редактора. До службових полів належать id, workspace_id, owner_id, developer_id, qa_reviewer_id, status, name, format, screen_format, created_at, updated_at, archived_at. До редакторських

даних належать layers, css_code, js_code, scene_config, timeline_steps, метадані ресурсів та пов’язані з експортом параметри [14].

Структуру основних таблиць інформаційної бази наведено у табл. 2.3.

Таблиця 2.3

Структура основних таблиць інформаційної бази

Таблиця	Основні поля	Призначення
profiles	id, email, full_name, avatar_url, created_at	Дані користувачів і зв’язок із Supabase Auth
workspaces	id, name, type, owner_id, plan, status	Опис персонального/командного робочого простору
workspace_members	workspace_id, user_id, role, workflow_role, status	Функціональна роль у команді
projects	id, workspace_id, name, status, format, layers, css_code, js_code	Дані рекламного проєкту та стан редактора
project_notifications	id, project_id, recipient_id, type, message, read_at	Повідомлення про перехід проєкту у інший статус, коментар QA і призначення
workspace_payments	id, workspace_id, provider, status, amount, invoice_id	Платіжна логіка активації робочого простору

У таблиці workspace_members важливо розділяти роль учасника і роль у робочому процесі. Роль учасника визначає, чи є користувач власником робочого простору або учасником. Поле workflow_role показує його функцію у виробничому процесі: розробник, QA або керівник. Такий поділ дає змогу гнучко керувати доступами: користувач може бути учасником, але виконувати роль QA; керівник може координувати задачі, не будучи власником; власник має адміністративні права на робочий простір.

2.4. Схема та фізична структура бази даних

Фізичну модель даних системи подано на рис. 2.2.

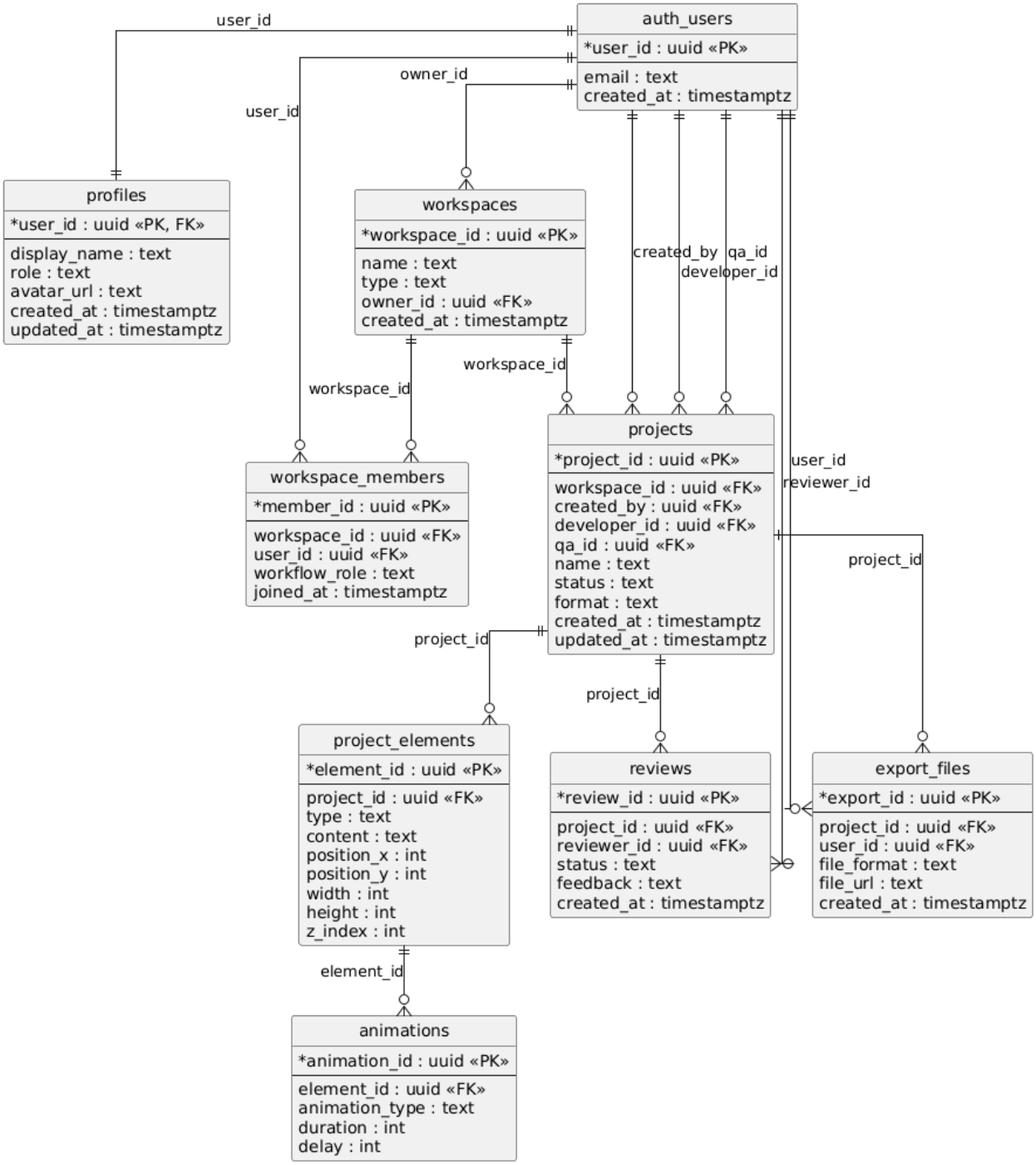

Рис. 2.2 Фізична модель даних системи

Фізична структура бази даних реалізується у Supabase PostgreSQL. Для забезпечення цілісності використовуються первинні ключі UUID, зовнішні ключі між таблицями, індекси за workspace_id, status, recipient_id, project_id і user_id, а також обмеження для допустимих значень статусів. Частина полів, які зберігають стан редактора, може бути подана у форматі JSONB, що дає змогу

гнучко зберігати шари, кроки часової шкали, параметри сцени та метадані ресурсів без надмірного дроблення схеми [4; 10; 14].

RLS-політики мають забезпечувати принцип мінімально необхідного доступу. Користувач повинен бачити лише ті проєкти, які належать робочому просторі, у якому він є учасником. Розробник може редагувати проєкт, якщо він призначений або має відповідні права в робочому просторі. QA може відкривати попередній перегляд і залишати коментар у межах доступного робочого простору. Власник і керівник можуть керувати призначеннями та переходами статусів відповідно до правил робочого процесу.

Приклад політик доступу на рівні рядків наведено на рис. 2.3.

```
-- Приклад логіки RLS для проєктів (спрощено)
create policy "members can view workspace projects"
on public.projects for select
to authenticated
using (
  exists (
    select 1 from public.workspace_members wm
    where wm.workspace_id = projects.workspace_id
      and wm.user_id = auth.uid()
      and wm.status = 'active'
  )
);

create policy "assigned developers can update project"
on public.projects for update
to authenticated
using (
  developer_id = auth.uid()
  or exists (
    select 1 from public.workspace_members wm
    where wm.workspace_id = projects.workspace_id
      and wm.user_id = auth.uid()
      and wm.workflow_role in ('lead')
  )
);
```

Рис 2.3 Приклад політик доступу на рівні рядків

Важливо, що RLS не замінює клієнтські перевірки, а доповнює їх. На рівні клієнтської частини користувач бачить лише доступні кнопки і дії, що покращує зручність взаємодії. На рівні бази даних RLS не дозволяє виконати операцію навіть у разі ручного API-запиту, якщо користувач не має належних прав. Такий подвійний контроль є базовою вимогою для системи з командними робочими просторами і ролями.

Під час проєктування інформаційного забезпечення важливо врахувати, що рекламний проєкт не існує сам по собі. Він завжди належить певному робочому простору, має автора або власника, може мати призначеного розробникаа та QA-рецензента, а також перебуває в одному зі статусів робочого процесу. Через це таблиця projects має бути пов’язана не тільки з користувачами, а й з таблицею workspaces. Такий зв’язок забезпечує групування проєктів за робочими просторами та дозволяє ізолювати дані різних команд.

Для workspace_members важливо зберігати не лише ідентифікатор користувача, а й роль у конкретному робочому просторі. Один і той самий користувач теоретично може бути власником в одному робочому просторі і розробником в іншому. Тому роль не повинна зберігатися тільки в профілі користувача. Вона є контекстною характеристикою участі у робочому просторі. Такий підхід дозволяє підтримувати персональні робочі простори, командні робочі простори, запрошення і запити на приєднання без змішування прав у різних командах.

У таблиці projects доцільно використовувати окремі поля для статусу, назви, опису, призначених користувачів і JSONB-структур редактора. Реляційні поля забезпечують фільтрацію, пошук, призначення та перевірки прав, тоді як JSONB дозволяє зберігати гнучкий стан редактора, який може містити сцени, шари, властивості елементів, CSS, JS, кроки часової шкали та інші параметри. Така комбінація робить модель достатньо структурованою для робочого процесу і водночас гнучкою для редактора креативів.

Повідомлення проєктів виконують роль журналу коротких подій, які мають бути видимими користувачам. До таких подій належать створення

проєкту, зміна статусу, передавання на QA, повернення з коментарем, погодження, запрошення до робочого простору або зміна призначень. На відміну від журналу аудиту, повідомлення орієнтоване на користувацький інтерфейс і повинна бути зручною для відображення у випадному списку або списку повідомлень. У майбутньому систему можна розширити повноцінним журналом аудиту.

workspace_payments необхідна для відокремлення бізнес-логіки активації командного робочого простору від звичайних даних про користувачів і проєкти. Платіж має мати власний статус, дату створення, час дії рахунку, ідентифікатор зовнішньої платіжної системи та зв’язок із робочим простором або користувачем. Це важливо для уникнення ситуацій, коли робочий простір активується без підтвердження платежу або коли повторний зворотний запит випадково змінює стан системи.

RLS-політики є критичним елементом інформаційної моделі. Якщо клієнтська частина перевіряє роль користувача тільки на рівні інтерфейсу, це не можна вважати достатнім захистом, тому що клієнтський код доступний користувачу. Реальна заборона доступу повинна виконуватися на рівні бази даних. Наприклад, користувач має бачити тільки ті projects, які належать робочому просторі, у якому він є учасником, а змінювати статуси можуть лише ролі з відповідними правами. Такий підхід узгоджує структуру даних із вимогами безпеки [14; 15].

Стан редактора є однією з найскладніших частин інформаційного забезпечення, тому що він описує не лише текстові поля, а повноцінний стан редактора. До нього можуть входити розміри сцени, фонові параметри, список шарів, тип кожного шару, координати, розміри, прозорість, обертання, текстовий вміст, шрифти, стилі, посилання на ресурси, параметри анімації та службові ознаки. Зберігати кожну дрібну властивість окремою таблицею можна, але для дипломного прототипу такий підхід був би надмірно складним [10; 14].

Використання JSONB дає змогу зберігати стан редактора як структурований документ, не відмовляючись від PostgreSQL. Перевага полягає в

тому, що модель може еволюціонувати разом із редактором: якщо додається новий тип шару або параметр анімації, не обов’язково щоразу змінювати схему бази даних. Водночас основні бізнес-поля, такі як статус, workspace_id, owner_id або assigned_qa_id, залишаються реляційними. Це поєднання підходів є практичним для вебредактора, де частина даних має жорстку структуру, а частина є гнучкою [10; 14].

Під час збереження стану редактора важливо уникати втрати даних при паралельних діях користувачів. Якщо кілька учасників відкривають один проєкт, система має або чітко обмежувати одночасне редагування, або передбачати механізм попередження про конфлікт. У поточній версії доцільно розглядати модель послідовного редагування з оновленням у реальному часі списку проєктів, а не повноцінну систему спільного редагування. Це спрощує реалізацію і зменшує ризики некоректного злиття змін [14].

Для майбутнього розвитку можна додати історію версій, де кожне суттєве збереження стану редактора створюватиме окрему версію. Це дозволить відновлювати попередні варіанти рекламного продукту, порівнювати зміни після коментаря QA і зменшувати ризик втрати результату. Однак така функція потребує додаткової структури даних, оптимізації зберігання великих JSON-документів і зрозумілого інтерфейсу для перегляду версій.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Абстракції предметної області

Абстракції предметної області дають змогу перейти від загального опису рекламного робочий процес до конкретних програмних об’єктів. Для Creative Workflow Studio ключовими абстракціями є користувач, робочий простір, рекламний проєкт, редактор, перевірка, рекламний продукт, коментар та експортований файл. Кожна абстракція має власні атрибути та поведінку, але вони взаємодіють у межах єдиного сценарію створення рекламного продукту [14; 15].

На рис. 3.1 наведено діаграму класів, яка відображає основні логічні об’єкти системи, їхні атрибути, методи та зв’язки між ними.

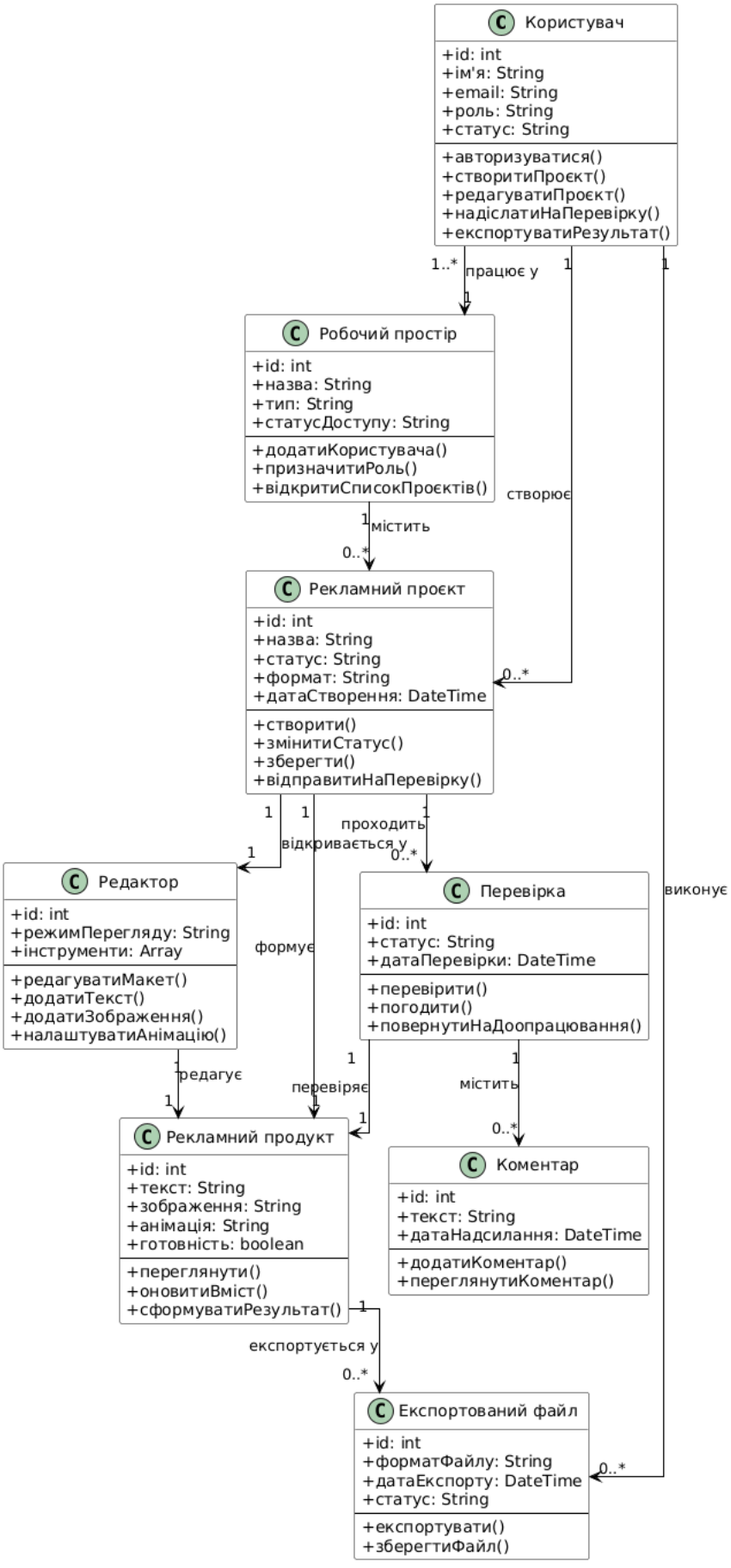

Рис. 3.1 Діаграма класів предметної області

Користувач має профіль і може бути учасником декількох робочих просторів. Робочий простір містить проєкти, учасників, права доступу й платіжний статус. Рекламний проєкт зберігає назву, формат, статус, призначення, дані редактора та історію взаємодії через повідомлення. Шар описує окремий текстовий або графічний елемент на робочому полотні. Крок анімації визначає, як і коли елемент має змінювати позицію, прозорість, масштаб або інші властивості [14].

У межах інтерфейсу ці об’єкти пов’язані з відповідними функціональними частинами системи: панель проєктів відображає список проєктів, редактор працює з активним проєктом і станом редактора, кабінет керує участю в робочому просторі, список повідомлень показує повідомлення про проєкти та учасників, а маршрути попереднього перегляду та експорту використовують збережені шари й дані анімації для відтворення результату.

3.2. Моделювання структури та взаємодії об’єктів

Пакетну структуру системи наведено на рис. 3.2.

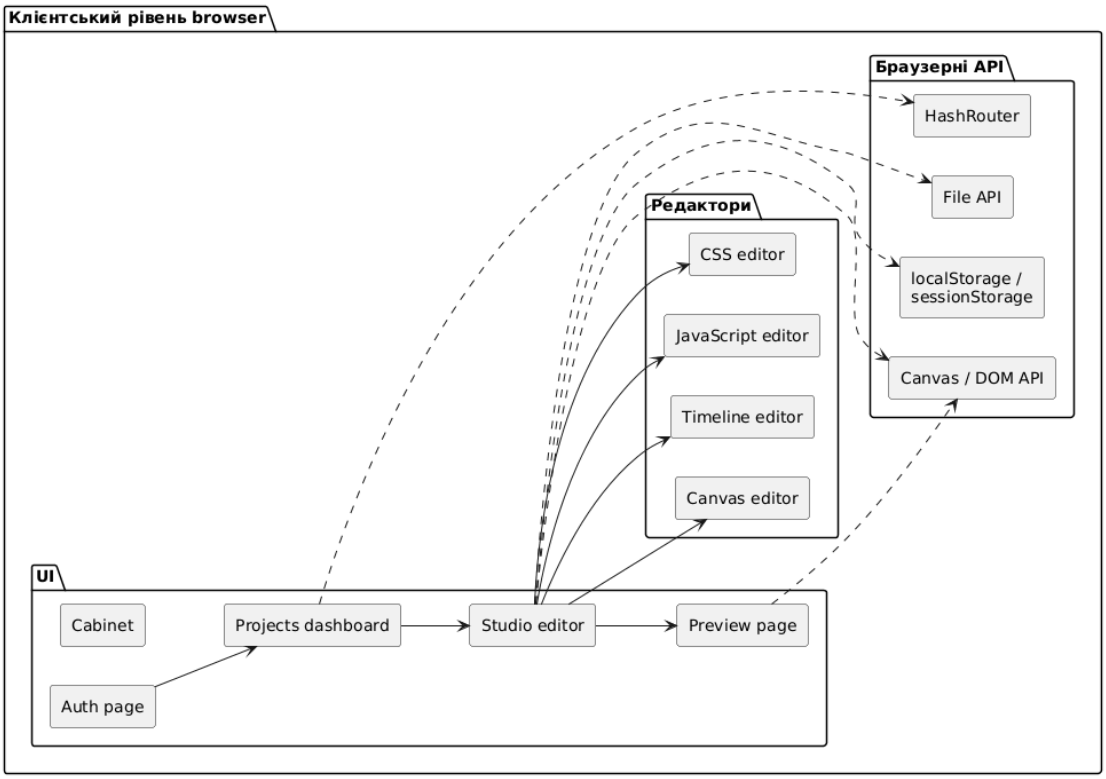

Рис. 3.2 Діаграма пакетів системи

Організація структури програмного забезпечення базується на розподілі функцій між пакетами та сервісами. Доцільно виділити пакет авторизації, пакет робочий простір-логіки, пакет панель проєктів, пакет редактор, пакет попередній перегляд/експорт, пакет повідомлення, пакет платіж та пакет Supabase services. Такий поділ спрощує розуміння системи і дає змогу поступово виносити логіку з великого App.jsx в окремі модулі, хуки та сервіси [14; 15].

Загальну компонентну взаємодію клієнтської частини та Supabase показано на рис. 3.3.

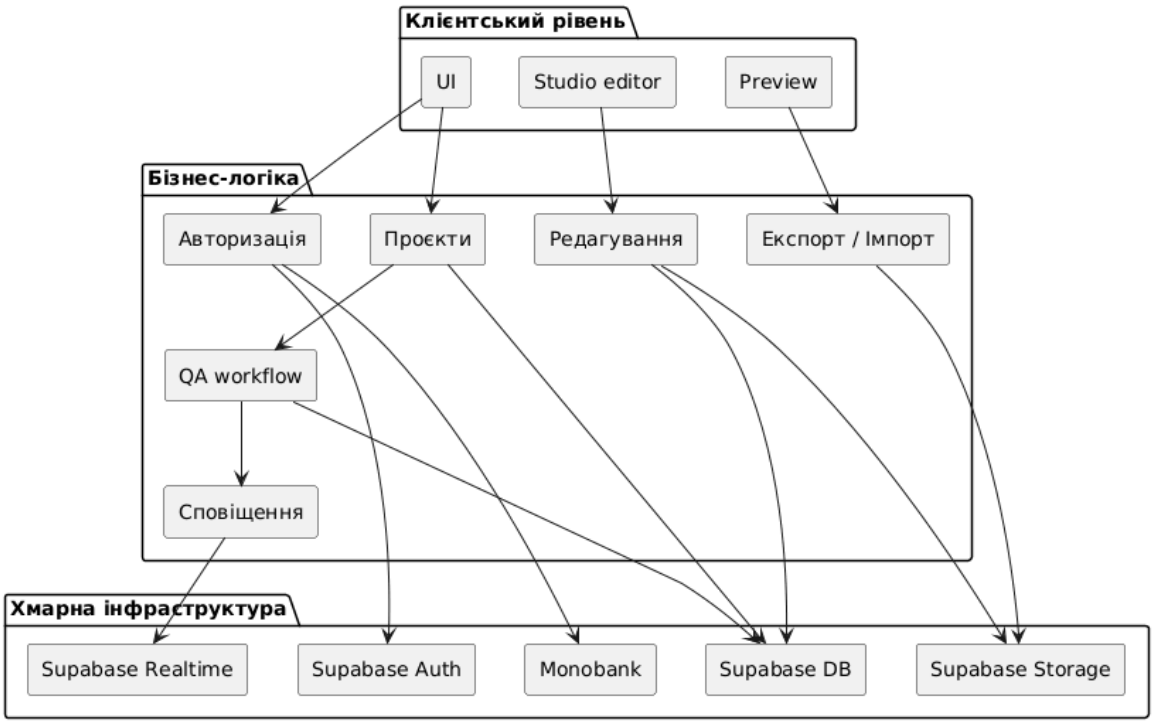

Рис. 3.3 Компонентна структура системи

Компонентна модель показує взаємодію клієнтської частини з Supabase. Клієнтський застосунок містить маршрути, компоненти інтерфейсу, стан редактора, сервіси й адаптери. Supabase забезпечує Auth, PostgreSQL, Realtime, RPC та Edge Functions. Окремо виділяється платіжний провайдер для створення рахунку і перевірки зворотного запиту. Така структура відповідає сучасному підходу до вебзастосунку, де значна частина інтерфейс працює в браузері, а модель «серверна частина як сервіс» (BaaS) надає керовані серверні можливості.

3.3. Організаційна структура програмного забезпечення

Основні маршрути вебзастосунку наведено у табл. 3.1.

Таблиця 3.1

Основні маршрути вебзастосунку

Маршрут	Призначення	Основна логіка
/auth	Реєстрація та вхід	Supabase Auth, гостьовий режим, перенаправлення після входу
/onboarding	Підключення робочого простору	Вибір персонального/командного плану, платіжний сценарій, активація
/projects	панель проєктів	Фільтри, статуси, дії відкрити/попередній перегляд/експорт/клонування /видалення
/cabinet	Кабінет користувача	Профіль, учасники робочого простору, ролі, запрошення
/projects/:id/editor	редактор Studio	Робоче полотно, шари, панель властивостей, JS/CSS/часова шкала
/projects/:id/preview	попередній перегляд	Відображення креативу в заданому форматі та орієнтації

Організаційна структура системи повинна підтримувати переходи між маршрутами без втрати контексту. Після перезавантаження сторінки застосунок має відновити аутентифікований маршрут, активний робочий простір і відкритий проєкт редактора. Це особливо важливо для прямого посилання на редактор або попередній перегляд, оскільки користувач може передати посилання іншому учаснику робочого простору або повернутися до задачі після перерви [3; 14; 17].

3.4. Компонентна структура системи

Деталізовану діаграму компонентів Creative Workflow Studio наведено на рис. 3.4.

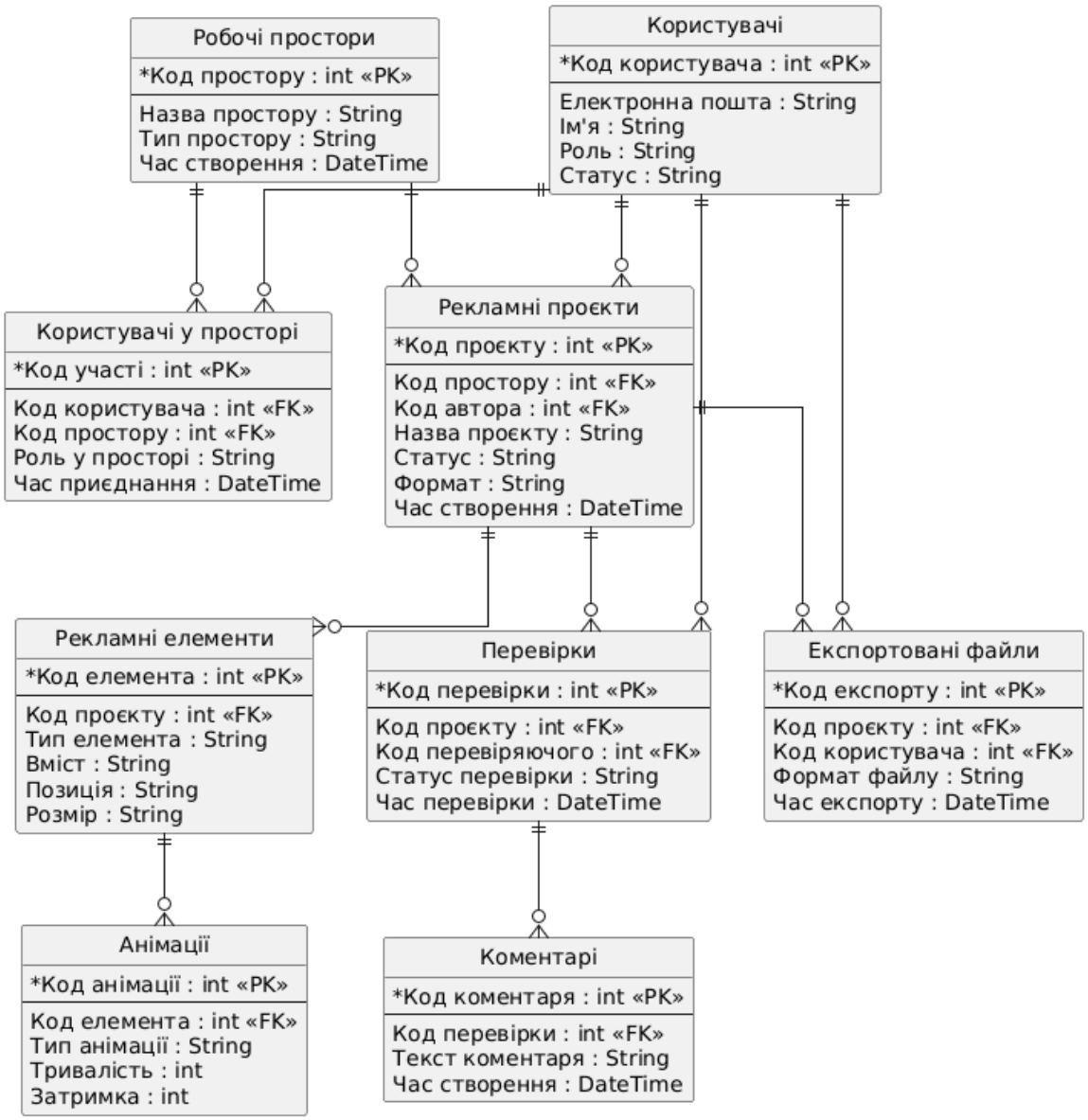

Рис. 3.4 Діаграма компонентів Creative Workflow Studio

На відміну від діаграми абстракцій, яка описує логічні об’єкти предметної області, компонентна діаграма показує фізично або функціонально відокремлені частини програмної системи та канали їхньої взаємодії. Компонентна структура Creative Workflow Studio розглядається на трьох рівнях: інтерфейс користувача, клієнтські сервіси та серверна частина Supabase. Рівень інтерфейсу відповідає за сторінки й елементи взаємодії, клієнтські сервіси виконують запити та обробку даних, а Supabase забезпечує збереження даних, політики доступу, серверні процедури, периферійні функції та оновлення в реальному часі.

Компоненти клієнтської частини програмного забезпечення наведено у табл. 3.2.

Таблиця 3.2

Компоненти клієнтської частини програмного забезпечення

Компонент	Функція	Дані / сервіси
AuthPage	Вхід і реєстрація	Supabase Auth, session state
ProjectsDashboard	Список проєктів і робочий процес	projects, filters, projectService
StudioEditor	Редагування документ креативу	шари, scene_config, css_code, js_code
TimelineEditor	Без кодова анімація	timeline_steps, ланцюжок GSAP
PreviewPage	Перегляд результату	project data, generated HTML/CSS/JS
ExportModal	Вибір формату експорту	HTML/GIF/MP4 generation pipeline
Cabinet	Профіль, команда, робочий простір	profiles, workspace_members, invites
Notifications	Повідомлення	project_notifications, Realtime subscription

3.5. Вибір інструментарію для створення прикладного програмного забезпечення

Для розробки інтерфейсу користувача веборієнтованого програмного забезпечення Creative Workflow Studio обрано стек React 18, Vite, JavaScript/TypeScript, CSS3, GSAP, Monaco Editor, Lucide React, Recharts та клієнт Supabase. Такий вибір зумовлений складністю інтерфейсу системи, оскільки застосунок є не просто набором сторінок, а повноцінною платформою робочого процесу для створення рекламного продукту: з редактором креативів, робочим полотном, шарами, інспектором властивостей, безкодовим редактором часової шкали, JavaScript/CSS-редакторами, попереднім переглядом, експортом, ролями користувачів, логікою робочого простору, повідомленнями та синхронізацією в реальному часі [1; 2; 3; 4; 5; 6; 7; 8; 11; 12; 14; 18].

Використаний технологічний стек системи узагальнено на рис. 3.5.

Рис. 3.5 Технологічний стек реалізації Creative Workflow Studio

Основним засобом побудови інтерфейсу обрано React 18, оскільки система містить велику кількість динамічних компонентів і станів: авторизація, активний робочий простір, список проєктів, фільтри, модальні вікна, статуси робочого процесу, стан редактора, повідомлення, стан платежу та доступи користувачів. На відміну від звичайного HTML/CSS/JavaScript-підходу, React дозволяє розділити інтерфейс на логічні компоненти, повторно використовувати блоки інтерфейсу та зручніше керувати змінами стану без ручного оновлення DOM. Це особливо важливо для редактора, де користувач постійно взаємодіє з шарами, властивостями елементів, робочим полотном, часова шкала та попередній перегляд [1; 2; 14].

Для збірки фронтенду використано Vite, оскільки він забезпечує швидкий запуск проєкту, зручний режим розробки та оптимізовану підсумкову збірку. Це відповідає потребам дипломного прототипу, де важливо швидко перевіряти зміни в інтерфейсі, тестувати редактор, модальні вікна, логіку робочого процесу та процес експорту й попереднього перегляду. Також Vite зручно поєднується з

React і дозволяє формувати готову збірку для розміщення застосунку, зокрема через GitHub Pages [3; 14].

У проєкті використовується змішаний підхід JavaScript і TypeScript. JavaScript застосовано для основної логіки інтерфейсу, зокрема в центральному компоненті застосунку, який керує авторизацією, робочим простором, проєктами, стан редактора, платежами та повідомленнями. TypeScript доцільний для більш структурованих частин системи, наприклад редактор часової шкали, типів анімацій, генерації GSAP-коду та сервісної логіки. Такий підхід дозволяє поєднати гнучкість JavaScript із більшою надійністю TypeScript у тих місцях, де важлива чітка структура даних [9; 14].

Для маршрутизації використано React Router із HashRouter. Це рішення є доцільним, оскільки застосунок має декілька основних маршрутів: авторизація, первинне налаштування, панель проєктів, кабінет, редактор та попередній перегляд. HashRouter дає змогу коректно відкривати сторінки за прямим посиланням без додаткового серверного резервного маршруту, що зручно для розгортання на статичному хостингу. Крім того, це дозволяє після перезавантаження сторінки відновлювати потрібний маршрут, активний робочий простір і поточний контекст редактора або попереднього перегляду [17; 14].

Для стилізації інтерфейсу використано CSS3. Власні стилі дають змогу точно контролювати зовнішній вигляд застосунку: таблиці проєктів, фільтри панелі інструментів, перемикач робочого простору, модальні вікна, статусні позначки, компонування редактора, плаваючі панелі, випадний список повідомлень та індикатор синхронізації в реальному часі. Використання CSS3 без надмірної залежності від важких бібліотек інтерфейсу дозволяє створити індивідуальний візуальний стиль продукту й адаптувати інтерфейс під специфіку виробничого процесу створення креативів, а не обмежуватися типовими готовими компонентами [9; 14].

Окремо обґрунтованим є використання GSAP для анімаційної логіки. Оскільки система орієнтована на створення цифрових креативів, анімація є

однією з ключових функцій продукту. GSAP дає змогу будувати точні анімації за часовою шкалою, керувати тривалістю, затримками, послідовністю та синхронізацією руху елементів. У проєкті це поєднано з безкодового редактора часової шкали: користувач може налаштовувати анімацію через інтерфейс, а система генерує ланцюжок GSAP, який потім використовується у попередній перегляд та процес експорту. Такий підхід робить систему корисною як для розробників, так і для дизайнерів або менеджерів виробництва, які не хочуть вручну писати JavaScript-код [7; 14].

Для реалізації вбудованого редактора коду обрано Monaco Editor. Це доцільно, оскільки Creative Workflow Studio містить JavaScript Editor і CSS Editor для досвідчених користувачів. Monaco Editor забезпечує звичний для розробників досвід редагування коду, подібний до середовища VS Code, і краще підходить для роботи з GSAP-логікою та CSS, ніж звичайне текстове поле. Це підвищує зручність роботи з кодом і робить систему більш професійною для користувачів, які хочуть мати не лише безкодові інструменти, а й можливість ручного доопрацювання [11; 14].

Для іконок використано Lucide React, оскільки ця бібліотека надає легкі SVG-іконки, які добре масштабуються, стилізуються через CSS і зручно інтегруються в React-компоненти. Іконки потрібні для меню, кнопок дій, елементи керування робочим простором, індикатор повідомлень, дії експорту, попереднього перегляду, відкриття й видалення та інших елементів інтерфейсу. SVG-формат дозволяє зберегти чіткість зображення на різних екранах і не перевантажує застосунок зайвими графічними ресурсами [18; 14].

Для відображення статистики в панель проєктів і кабінет доцільно використано Recharts. У системі є блоки з метриками проєктів, статуси розроблення, QA-перевірка, готовність до використання, кількість призначених задач і загальна активність робочого простору. Recharts дозволяє візуалізувати ці дані у зрозумілій формі та органічно інтегрується з React. Це робить інтерфейс не лише функціональним, а й більш інформативним для власник, керівник, розробник і QA-користувачів [12; 14].

Для взаємодії з серверною частиною використано клієнт Supabase, оскільки інтерфейс напряму пов’язаний з авторизацією, логікою робочого простору, ролями, проєктами, повідомленнями та оновленнями в реальному часі. клієнт Supabase дозволяє отримувати дані про користувача, робочий простір, список проєктів, статуси, повідомлення та участь у робочому просторі, а також підтримувати синхронізацію в реальному часі між учасниками команди. Це важливо для системи, де зміни статусу, призначення розробника/QA або нові повідомлення мають оперативно відображатися в інтерфейсі [4; 5; 6; 8; 14].

Вибір саме компонентного клієнтська частина-стеку є виправданим через складність редактор Studio. Інтерфейс редактора включає верхню панель меню, робоче полотно, панель властивостей, панель документа, плаваюче вікно шарів, JS редактор, CSS редактор, редактор часової шкали і службові модальні вікна. У межах такого інтерфейсу користувач може додавати й редагувати шари, змінювати позицію, розмір, прозорість, обертання, текстові властивості, порядок елементів, а також налаштовувати анімацію та експортувати результат. Реалізація такої кількості взаємопов’язаних елементів інтерфейсу без React була б менш масштабованою та складнішою для підтримки [14].

Також важливо, що обраний стек підтримує подальше розширення системи. У майбутньому можна винести частину логіки з великого App.jsx у власні хуки або окремі модулі стану, додати більше рольових E2E-сценаріїв, журналу аудиту, історію версій, стрічку активності і розширений набір шаблонів. Тобто React/Vite/Supabase/GSAP-архітектура не лише відповідає поточній версії застосунку, а й залишає можливість для поступового розвитку продукту [1; 2; 3; 4; 5; 6; 7; 8; 14].

Отже, вибір засобів розробки інтерфейсу користувача є обґрунтованим функціональними вимогами Creative Workflow Studio. React забезпечує компонентність і керування складним станом інтерфейсу, Vite - швидку розробку та збірку, CSS3 - гнучке оформлення інтерфейсу, GSAP - професійну анімаційну логіку, Monaco Editor - зручну роботу з кодом, Lucide React - легкі SVG-іконки, Recharts - візуалізацію статистики, а клієнт Supabase - зв’язок інтерфейсу з

авторизацією, даними, ролями та синхронізацією в реальному часі. У сукупності ці засоби відповідають меті проєкту – створити веб платформу для повного циклу роботи з рекламними креативами: від створення й анімації до QA-перевірки, командної взаємодії, експорту та передачі у робоче середовище [1; 2; 3; 4; 5; 6; 7; 8; 11; 12; 14; 18].

React обрано через компонентний підхід і можливість керувати складним станом інтерфейсу. Офіційна документація React описує стан як механізм, який дозволяє компоненту «пам’ятати» значення між взаємодіями користувача, а хуки – як спосіб використовувати функції React у компонентах [1; 2; 14]. Для Creative Workflow Studio це важливо, оскільки застосунок має багато станів: активний робочий простір, поточний користувач, список проєктів, вибраний шар, стан редактора, стан повідомлень, стан модального вікна і стан платежу.

Vite використовується як інструмент розробки та збірки. Для дипломного прототипу важливими є швидкий сервер, зручна інтеграція з React і підсумкова збірка, який можна розміщувати на статичному хостингу [3; 14]. Це відповідає вимозі демонстраційного розгортання веб додатку поза локальним комп’ютером.

Supabase обрано як платформу серверної частини. Supabase Auth забезпечує автентифікацію та інтегрується з даними через JWT і RLS [8; 4]. Realtime дозволяє підписуватися на зміни в таблицях PostgreSQL і оновлювати інтерфейс без ручного перезавантаження [5; 6]. Edge Functions дають змогу реалізовувати серверні TypeScript-функції для вебхуків, платежів і складніших операцій [6].

GSAP використовується для анімації рекламних елементів. Часова шкала у GSAP є інструментом послідовного керування анімаціями, що дозволяє створювати складні animation sequences без ручного розрахунку затримок для кожного tween [7; 14]. У системі цей підхід поєднано з безкодового редактора часової шкали, де користувач налаштовує анімацію в інтерфейсі, а програма генерує відповідний ланцюжок GSAP.

3.6. Алгоритмізація та програмування програмних модулів

Алгоритмізація програмних модулів Creative Workflow Studio пов’язана з основними сценаріями: створення нового рекламного проєкту, передавання креативу на QA, повернення з коментарем, переведення у стан готовності до використання, експорт результату, приєднання до робочого простору, активація платежу та оновлення панелі проєктів у реальному часі. У межах роботи детально розглянуто три базові алгоритми: створення проєкту, передавання на QA та експорт [14; 15]. Алгоритм створення рекламного проєкту подано на рис. 3.6.

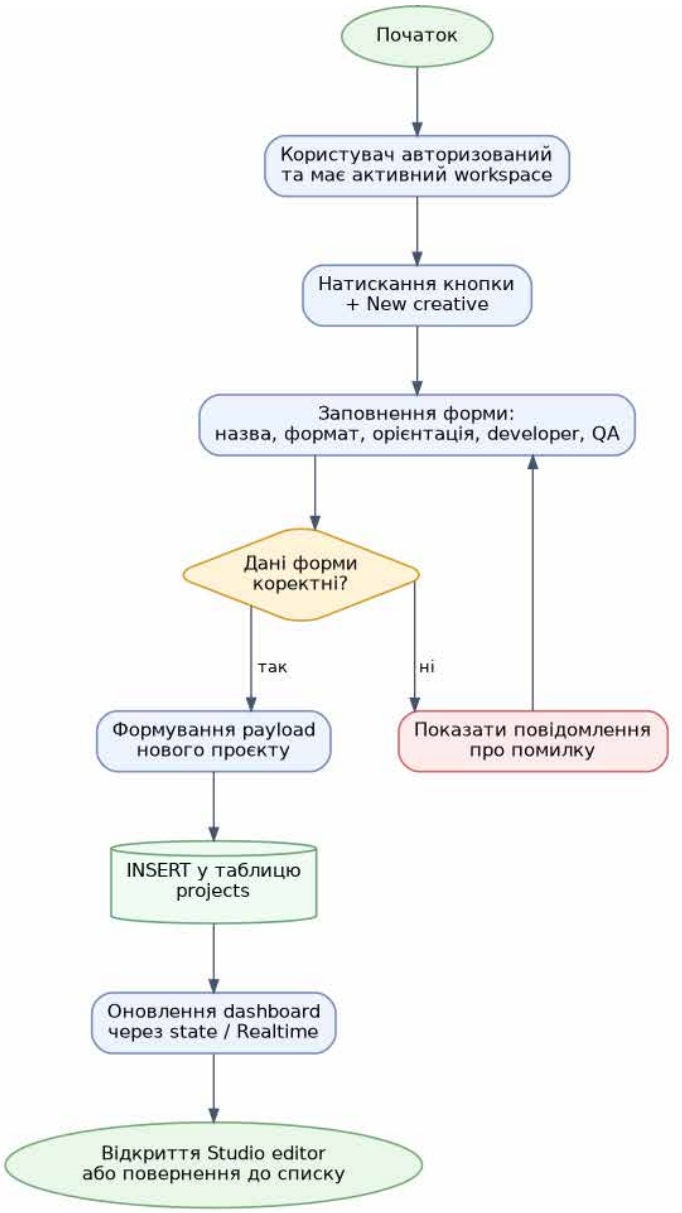

Рис. 3.6 Алгоритм створення рекламного проєкту

Алгоритм створення рекламного проєкту починається з перевірки активного робочий простір і авторизованого користувача. Далі система перевіряє форму, формує набір даних, записує дані в таблицю projects, задає початковий статус розроблення, створює базові параметри сцени й оновлює список проєктів. Якщо дані некоректні або робочий простір недоступний, користувач отримує повідомлення про помилку [4; 10; 14; 15].

Фрагмент створення рекламного проєкту на рівні клієнтської частини наведено на рис 3.7.

```
async function handleCreateProject(formValues) {
  if (!activeWorkspace?.id || !currentUser?.id) {
    showToast('Workspace is not available');
    return;
  }

  const name = String(formValues.name || '').trim();
  if (!name) {
    showToast('Project name is required');
    return;
  }

  const payload = {
    workspace_id: activeWorkspace.id,
    owner_id: currentUser.id,
    name,
    status: 'Development',
    format: formValues.format || '300x250',
    screen_format: formValues.orientation || 'landscape',
    developer_id: formValues.developerId || currentUser.id,
    qa_id: formValues.qaId || null,
    images: [],
    code: '',
    css_code: '',
    scene_background: '#ffffff',
    scene_border_style: 'none',
    scene_border_color: '#000000',
  };

  const createdProject = await projectService.createProject(payload);
  setProjects((prev) => [createdProject, ...prev]);

  if (formValues.openAfterCreate) {
```

```
        navigate(`/projects/${createdProject.id}/editor`);
    }
}
```

Рис. 3.7 Створення рекламного проєкту на рівні клієнтської частини

Алгоритм передавання креативу на QA-перевірку наведено на рис. 3.8.

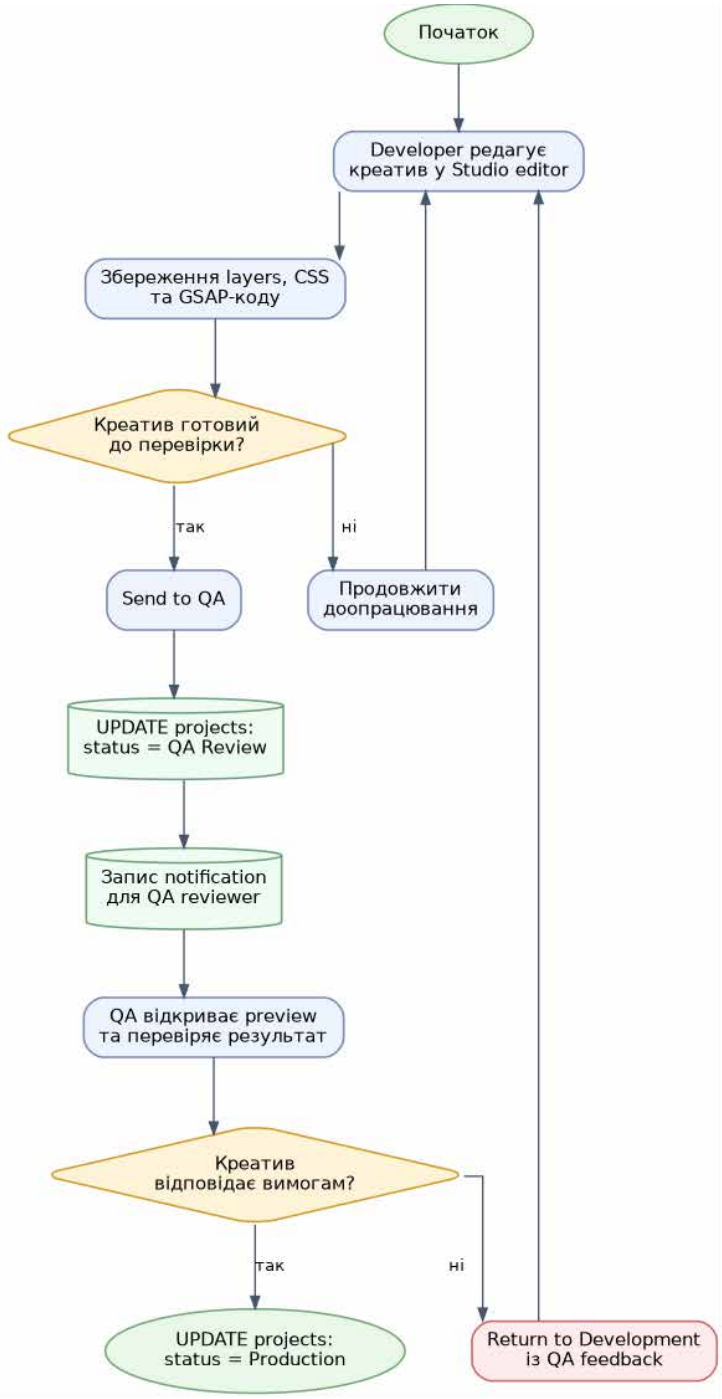

Рис. 3.8 Алгоритм передавання креативу на QA-перевірку

Передавання на QA є перехід робочого процесу. Перед зміною статусу система повинна переконатися, що проєкт існує, користувач має право виконувати дію, призначено QA-рецензента, стан редактора збережено, а супровідну примітку або службові дані зафіксовано. Після зміни статусу створюється повідомлення для QA і панель проєктів оновлюється через у реальному часі-механізм [4; 10; 14; 15].

Фрагмент передавання проєкту на QA-перевірку наведено на рис. 3.9.

```
async function handleSendToQa(project, handoffNote) {
  if (!project?.id) {
    return;
  }

  if (!project.qa_id) {
    showToast('QA reviewer is not assigned');
    return;
  }

  await projectService.saveProject(project.id, {
    images: editorLayers,
    code: jsCode,
    css_code: cssCode,
    scene_background: sceneBackground,
    scene_border_style: sceneBorderStyle,
    scene_border_color: sceneBorderColor,
  });

  const updatedProject = await projectService.updateProjectStatus({
    projectId: project.id,
    nextStatus: 'QA Review',
    note: handoffNote || 'Creative is ready for QA review',
  });

  setProjects((prev) =>
    prev.map((item) => item.id === updatedProject.id ? updatedProject : item)
  );
  showToast('Project was sent to QA');
}
```

Рис. 3.9 Передавання проєкту на QA-перевірку

Алгоритм експорту рекламного продукту наведено на рис. 3.10.

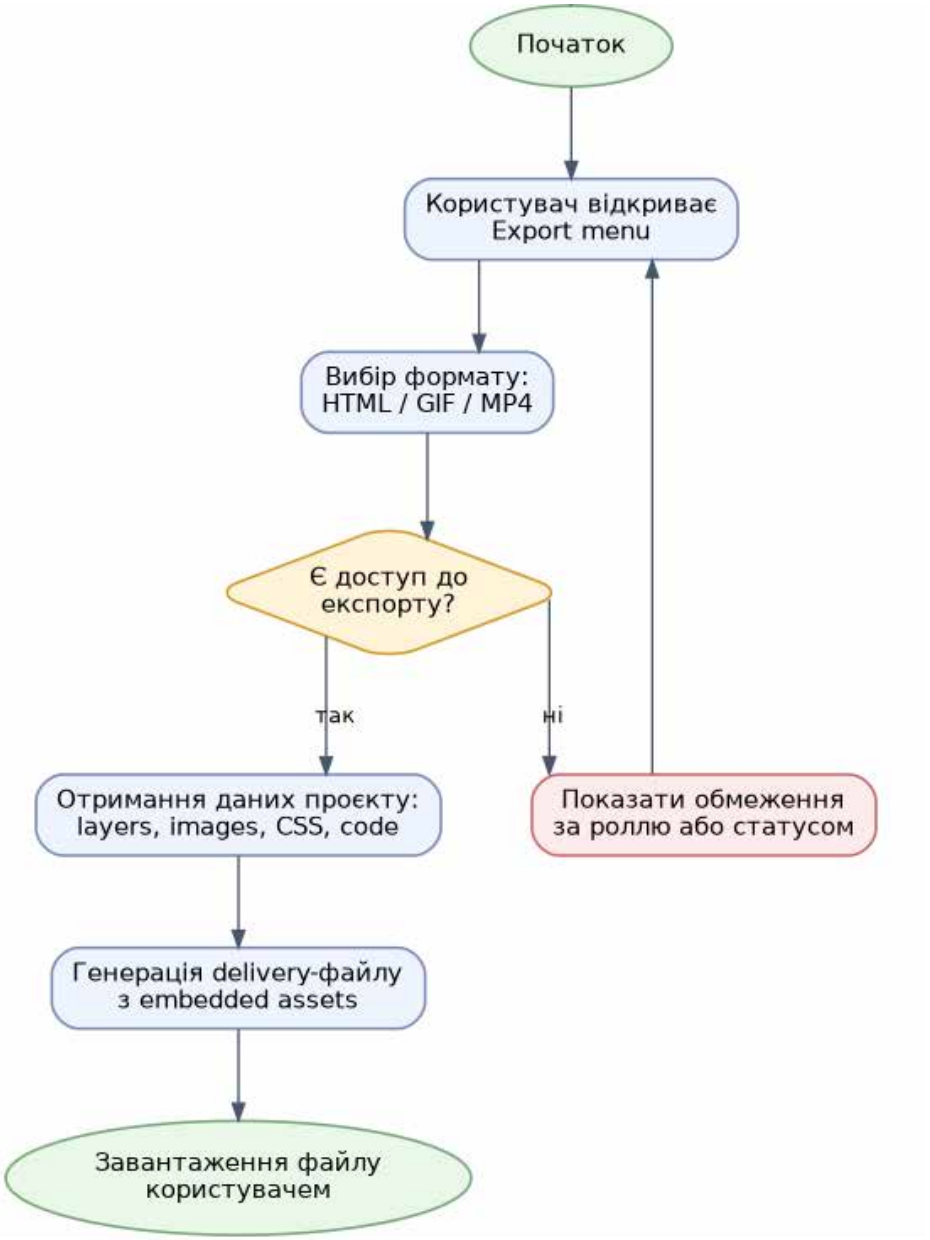

Рис. 3.10 Алгоритм експорту рекламного продукту

Експорт рекламного продукту повинен враховувати обраний формат. HTML-експорт формує автономний файл із розміткою, стилями, ресурсами та скриптом анімації. GIF/MP4-експорт використовується для демонстрації або передачі матеріалу попереднього перегляду. Перед експортом система має перевірити доступ користувача до проєкту, актуальність збережених даних і технічні обмеження формату [7; 9; 14; 15].

Фрагмент підписки на зміни у робочому просторі наведено на рис. 3.11.

```
useEffect(() => {
    if (!activeWorkspace?.id) {
        return;
    }

    const channel = supabase
        .channel(`workspace-projects-${activeWorkspace.id}`)
        .on(
            'postgres_changes',
            {
                event: '*',
                schema: 'public',
                table: 'projects',
                filter: `workspace_id=eq.${activeWorkspace.id}`,
            },
            () => reloadProjects(activeWorkspace.id)
        )
        .subscribe();

    return () => {
        supabase.removeChannel(channel);
    };
}, [activeWorkspace?.id]);
```

Рис. 3.11 Realtime-підписка на зміни у робочий простір

Головна сторінка зі списком рекламних проєктів показана на рис. 3.12.

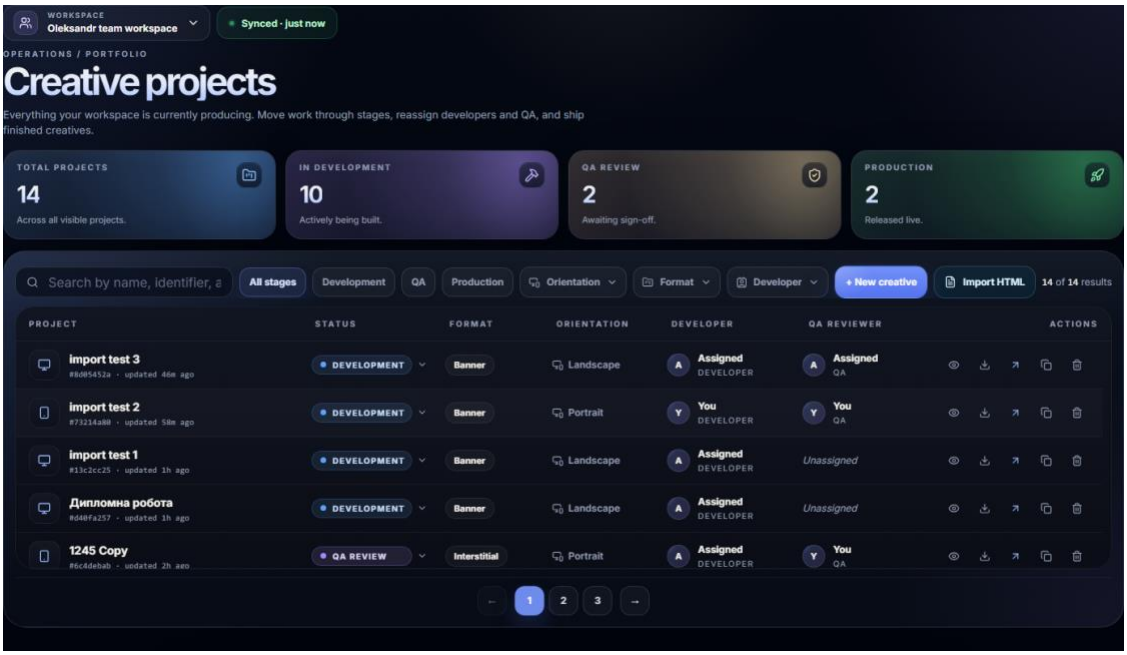

Рис. 3.12 Головна сторінка зі списком рекламних проєктів

Інтерфейс редактора рекламного продукту показано на рис. 3.13.

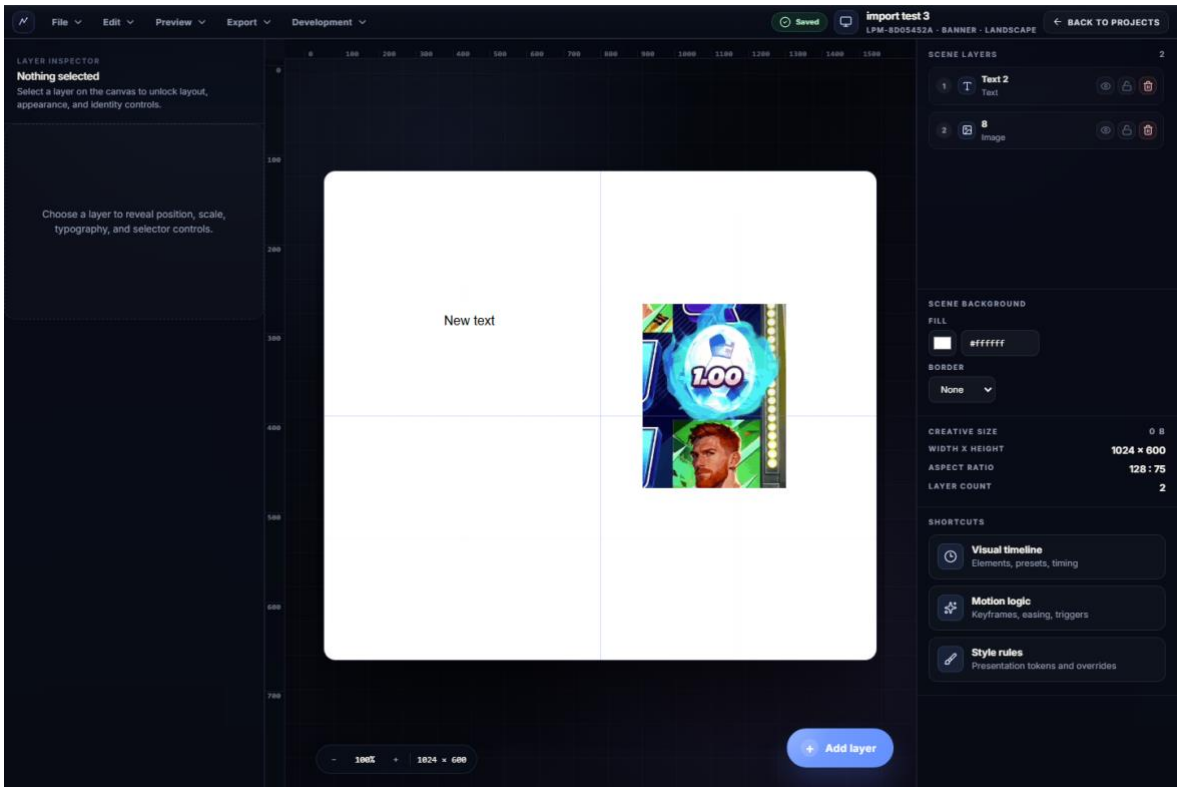

Рис. 3.13 Редактор Studio для редагування рекламного продукту

Наведені фрагменти показують, що основна логіка системи реалізується на стику стану клієнтської частини, клієнт Supabase, SQL-структури і RLS/RPC-перевірок. Такий підхід дає змогу швидко реалізувати повний робочий процес, але вимагає уважного розділення відповідальності: інтерфейс не повинен бути єдиним бар'єром безпеки, а серверні процедури мають перевіряти права користувача перед критичними змінами [4; 5; 10; 14].

Клієнтська частина Creative Workflow Studio має складнішу структуру, ніж звичайний інформаційний сайт, оскільки вона поєднує кілька режимів роботи: авторизацію, первинне налаштування, панель проєктів, кабінет, редактор, попередній перегляд і експорт. Кожен із цих режимів має власний набір станів, але вони не є повністю незалежними. Наприклад, активний робочий простір впливає на список проєктів, доступність кнопок, склад команди, список повідомлень, можливість створення нового креативу та право переходу до

редактор. Тому app-level state має координувати глобальний контекст користувача [1; 2; 14].

Сторінку створення облікового запису показано на рис. 3.14.

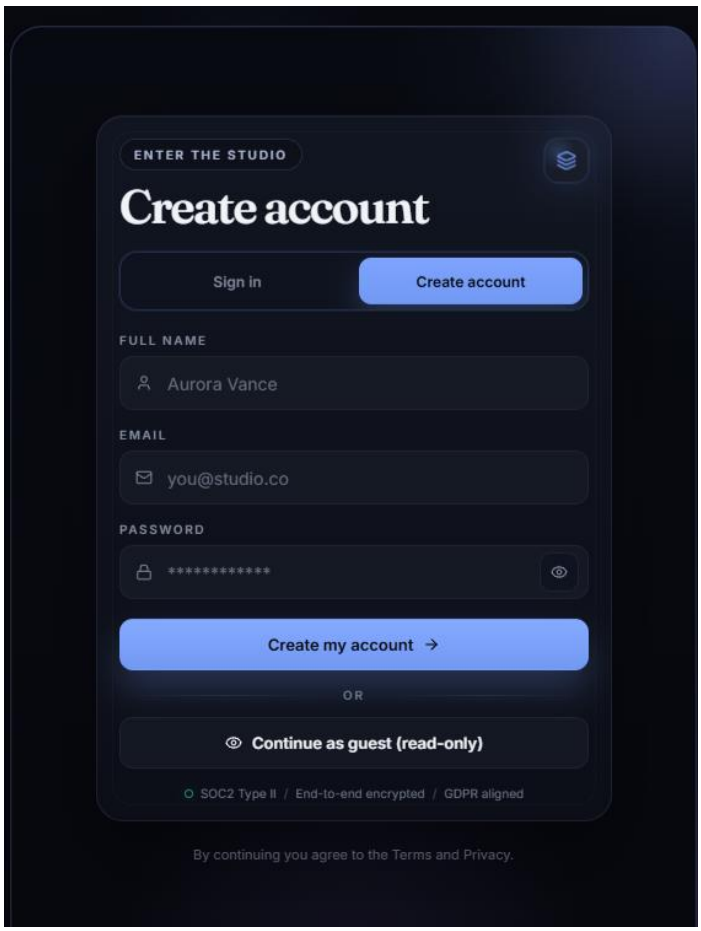

Рис. 3.14 Сторінка створення облікового запису

Стан авторизації формується після отримання сесію від SupabaseAuth. Після цього застосунок повинен завантажити профіль, визначити доступні робочі простори, вибрати активний робочий простір і тільки потім відкривати основні сторінки. Якщо користувач перезавантажує сторінку на пряме посилання, наприклад /projects/, система не повинна автоматично перекидати його на початковий екран. Спочатку необхідно відновити сесія, завантажити робочий простір context, перевірити доступ до проєкту і лише після цього відкрити редактор або показати повідомлення про відсутність прав [8; 14].

Панель проєктів, що відображає поточний стан робочого простору, наведено на рис. 3.15.

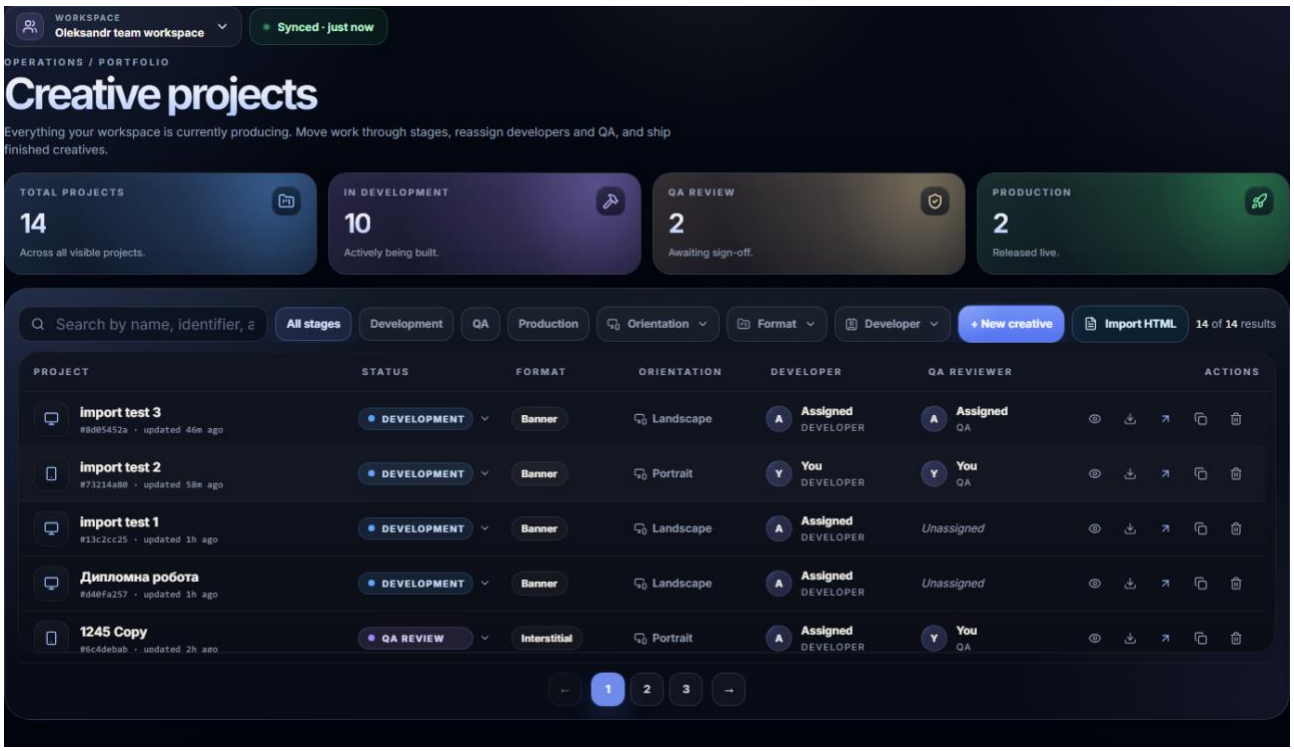

Рис. 3.15 Панель проєктів і керування рекламними креативами

Екран редактора Studio наведено на рис. 3.16.

Панель проєктів є центральним екраном управління рекламними проєктами. Вона повинна одночасно показувати список креативів, статуси робочого процесу, призначеного розробникаа, призначеного QA, фільтри, кнопки попереднього перегляду, експорту, відкриття, клонування й видалення, а також поточні обмеження ролі. З технічної точки зору це вимагає узгодження даних із таблиці projects, profiles, workspace_members і сервісу повідомлень. Для зручності користувача частина перевірок виконується на рівні клієнтської частини, але критичні операції мають дублюватися на рівні RLS або RPC [14].

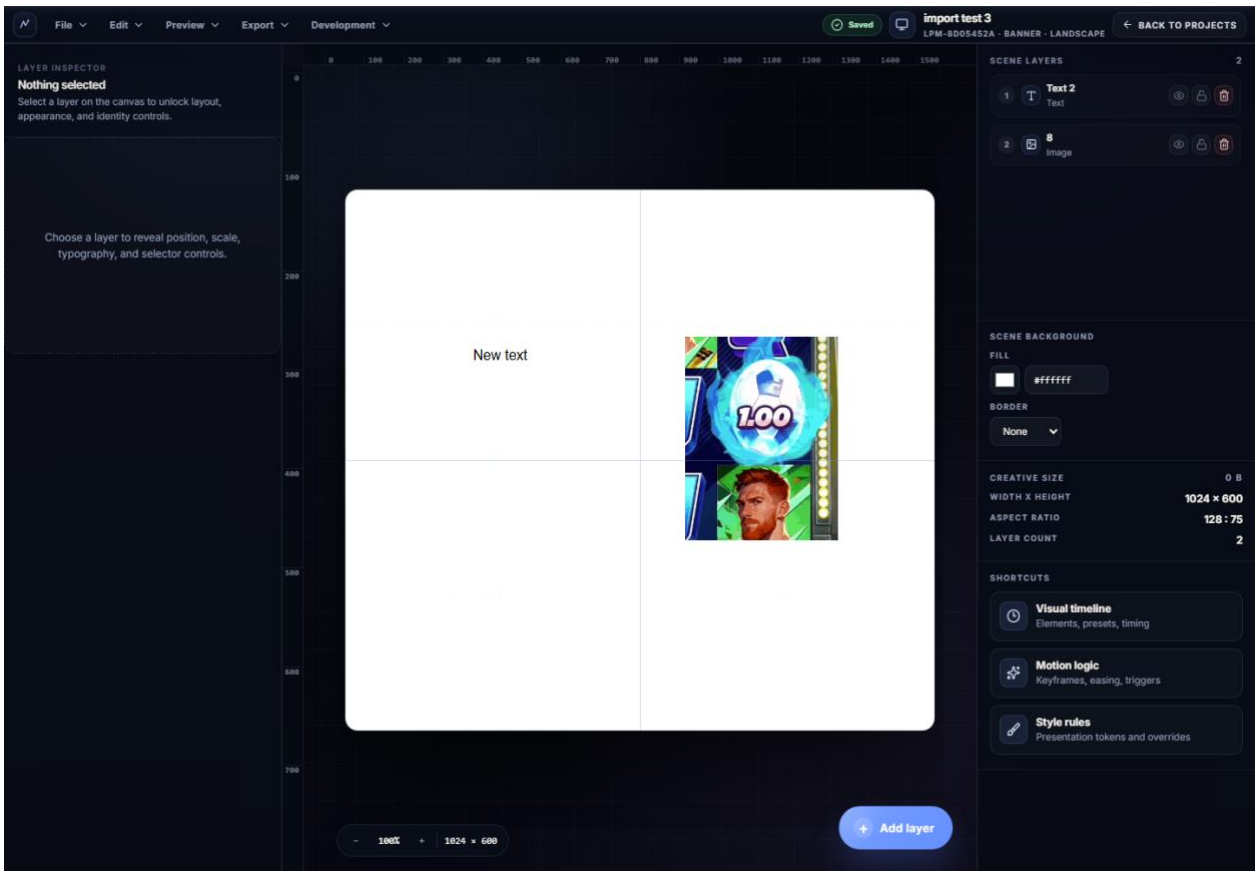

Рис. 3.16 Редактор Studio для редагування рекламного продукту

Редактор Studio має власну внутрішню модель, яка відрізняється від простого списку полів форми. У ньому є вибраний шар, масштаб робочого полотна, стан панелі властивостей, налаштування документа, CSS/JS-код, кроки часової шкали, стан попереднього перегляду, стан експорту і службові модальні вікна. Зміна одного елемента може впливати на кілька частин інтерфейсу: наприклад, вибір шару оновлює панель властивостей, панель шарів і доступні дії часової шкали. Тому редактор потребує чіткої структури стану та обережного оновлення, щоб уникнути неузгодженості між відображенням і збереженими даними [1; 2; 7; 11; 14].

Панель шарів виконує роль навігації по структурі документ креативу. Користувач повинен бачити текстові й графічні елементи, мати змогу вибрати активний шар, змінити порядок, видалити або редагувати властивості. Для рекламного продукту порядок шарів має практичне значення, тому що він визначає перекриття елементів на робочому полотні. Збереження порядку шарів

у стан редактора дозволяє відтворити макет при відкритті проєкту в іншому браузері або після перезавантаження [14].

Панель властивостей вибраного елемента наведено на рис. 3.17.

Панель властивостей деталізує властивості вибраного елемента. Він повинен змінювати набір полів залежно від типу шару: для тексту важливі вміст, розмір шрифту, колір, вирівнювання; для зображення - ресурс, розміри, позиція, припасування об’єкта; для будь-якого шару - координати, прозорість, обертання і видимість. Така логіка показує перевагу компонентного підходу React: різні підкомпоненти панелі властивостей можуть відповідати за окремі групи властивостей, а зміни передаються до спільного стану редактора [14].

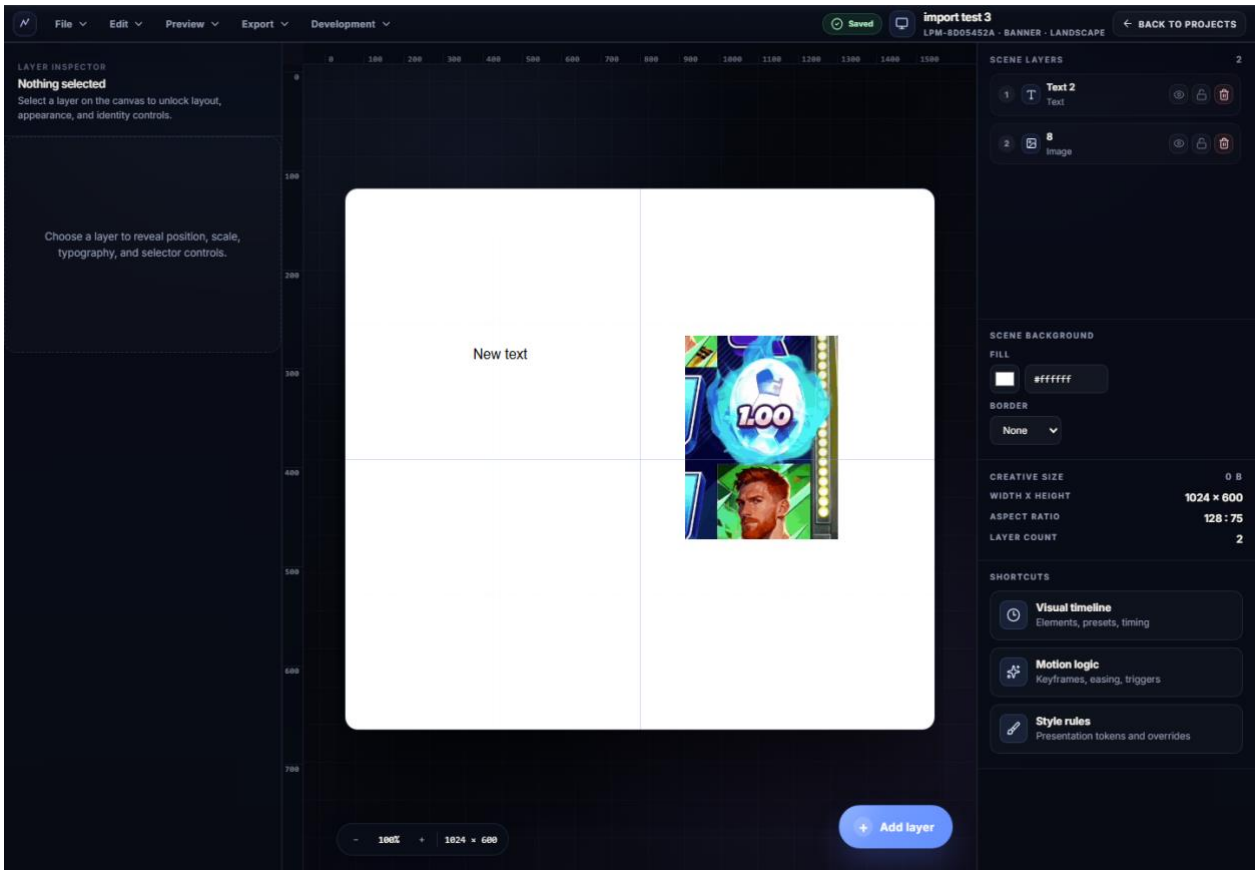

Рис. 3.17 Панель властивостей вибраного елемента

Безкодовий редактор часової шкали анімації наведено на рис. 3.18.

Безкодовий редактор часової шкали є функцією, що поєднує налаштування інтерфейсу з генерацією анімаційної логіки. Користувач не пише GSAP-код вручну, а вибирає елемент і задає параметри анімації. Система перетворює ці

налаштування на послідовність кроків часової шкали, які потім можуть бути представлені як ланцюжок GSAP. Такий механізм важливий для нетехнічних користувачів, але він також має бути передбачуваним для розробника, який може перевірити або змінити згенерований JavaScript [7; 14].

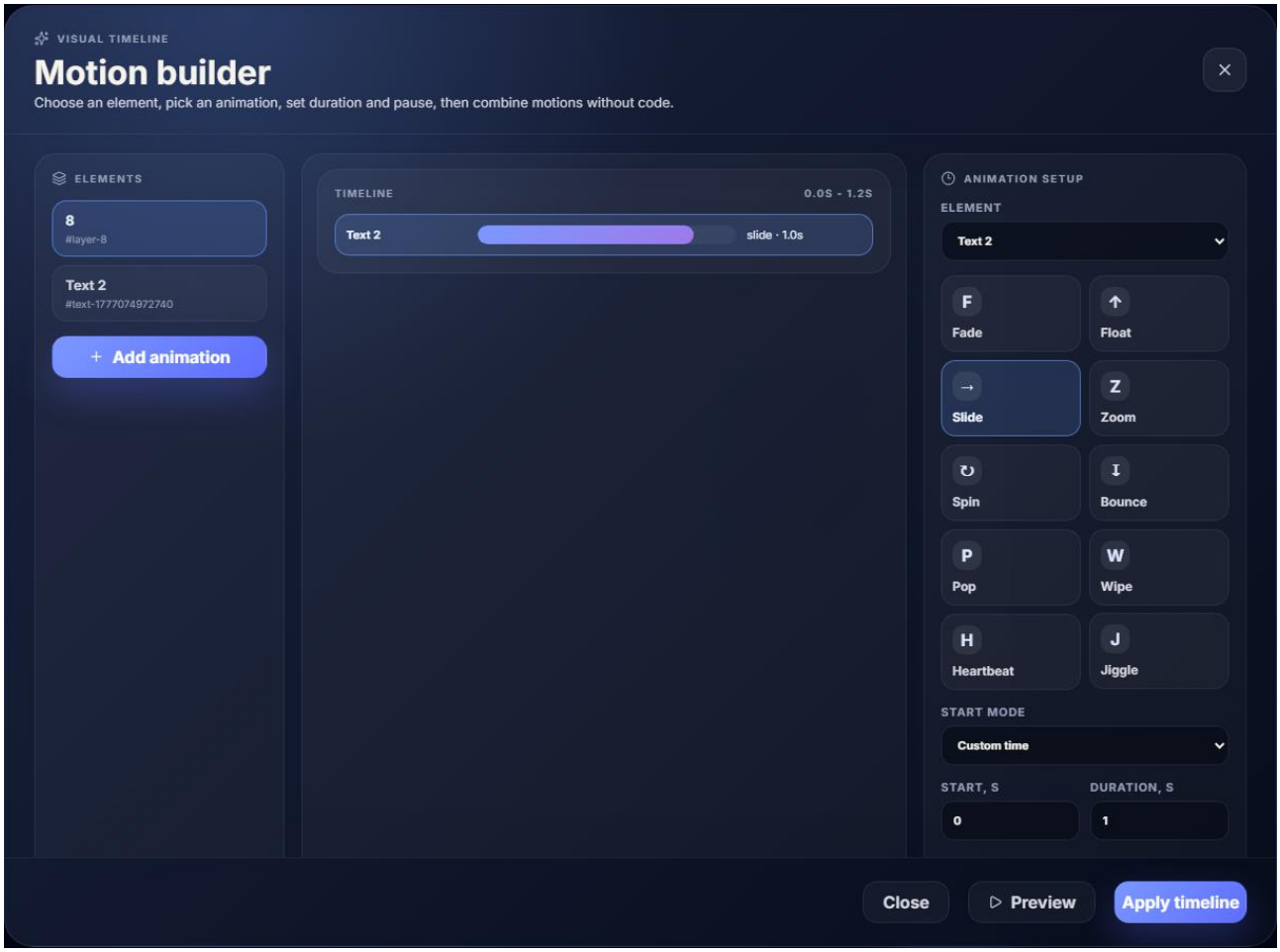

Рис. 3.18 Безкодовий редактор часової шкали анімації

Редактор JavaScript і CSS для ручного доопрацювання креативу наведено на рис. 3.19.

JavaScript і CSS редактор реалізують принцип code escape hatch. Це означає, що система не примушує просунутого-користувача обмежуватися тільки безкодовий інструментами. Якщо потрібна складніша анімація, нестандартна інтерактивність або спеціальні стилі, користувач може внести код вручну. Водночас така можливість підвищує вимоги до безпеки попередній перегляд та експорт, оскільки виконання довільного HTML/JS може створювати

ризики. Тому HTML імпорт і попередній перегляд потребують обережного обмеження й перевірки [11; 14].

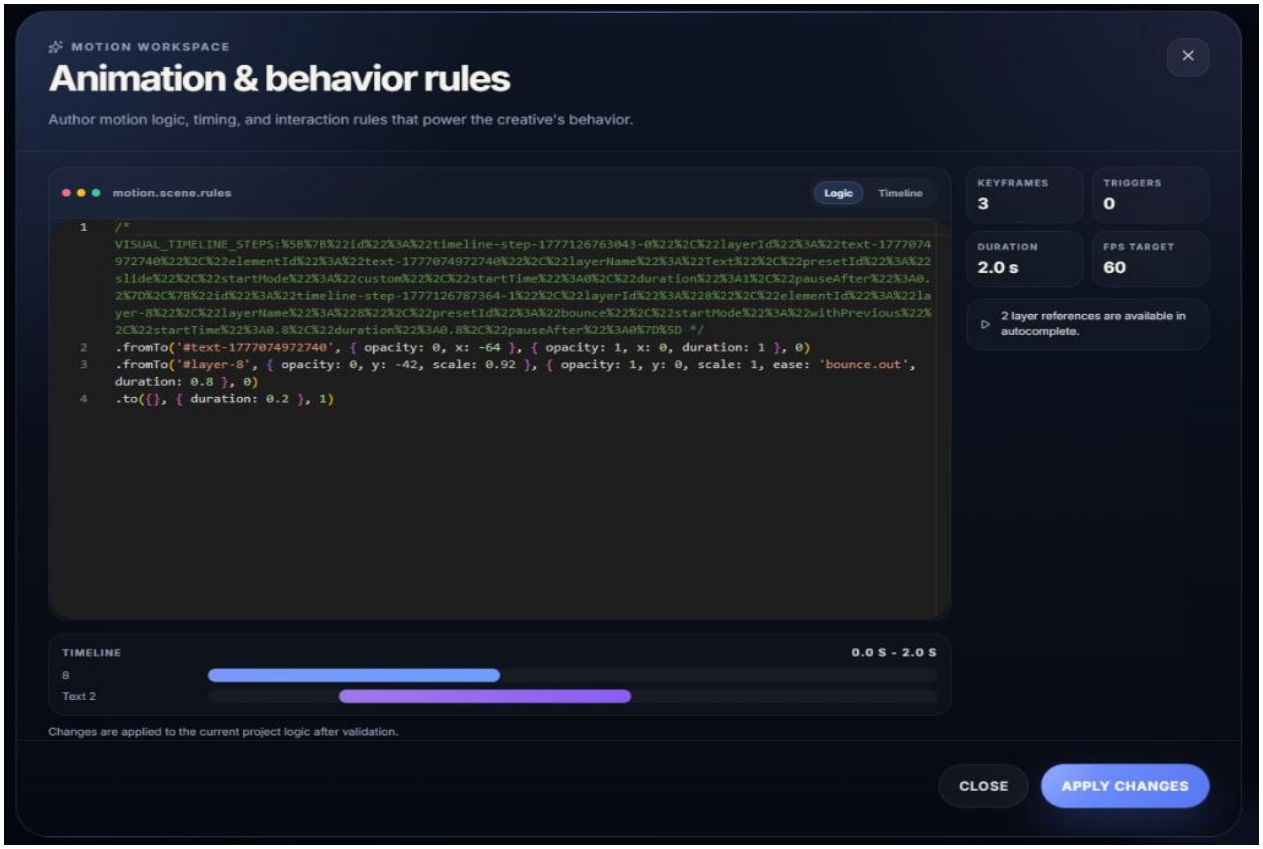

Рис. 3.19 Редактор JavaScript і CSS для ручного доопрацювання креативу

Алгоритм створення проєкту починається ще до запису рядка в базу даних. Спочатку перевіряється, чи користувач авторизований і чи має активний робочий простір. Далі система визначає тип робочого простору, роль користувача, доступність плану, валідність назви проєкту та початкові параметри креативу. Тільки після цього формується дані запиту для таблиці projects. Така послідовність дозволяє уникнути створення неповних або «сирих» проєктів, які неможливо коректно відкрити в редакторі [4; 8; 10; 14].

Під час збереження проєкту важливо розрізняти два типи даних: службові бізнес-поля і власне документ креативу. Бізнес-поля потрібні для панелі проєктів, фільтрації та доступів; документ креативу потрібний для редактора й експорту. Якщо ці дані змішати без структури, система стане складною для підтримки. Тому логічно мати окремі поля для status, title, workspace_id,

assigned_developer_id, assigned_qa_id і окремі JSONB-поля для scene_config, layers, css_code, js_code та timeline_data [10; 14].

Передавання проєкту на QA-перевірку не повинно бути простим оновленням текстового статусу. Це перехід робочого процесу, який має перевірити попередній статус, роль користувача, наявність призначеного QA, збереження останніх змін і можливість створити повідомлення. Якщо хоча б одна умова не виконується, система повинна показати зрозуміле повідомлення і не змінювати стан проєкту. Такий алгоритм забезпечує цілісність процесу та зменшує ймовірність помилок у командній роботі [4; 10; 14].

Повернення з коментар QA є зворотним переходом, який має зберігати причину повернення. Зворотний зв’язок не повинен губитися після оновлення сторінки, тому його потрібно фіксувати в даних проєкту або окремій таблиці повідомлень. Для розробник важливо бачити не тільки факт повернення, а й конкретний текст зауваження. Для керівник або власник така інформація є показником якості процесу: вона дозволяє відстежувати, які типові помилки повторюються і де потрібно покращити шаблони або інструкції [14].

Затвердження та готовність до використання є критичною дією, тому що після неї рекламний продукт вважається готовим. Система має обмежити виконання цієї дії ролями QA, керівник або власник залежно від обраної політики. Розробник може підготувати проєкт, але остаточне погодження має виконувати користувач, відповідальний за контроль якості. У майбутньому статус готовності до використання можна пов’язати з додатковим блокуванням редагування або створенням фінального експорт файлу [4; 10; 14].

Алгоритм експорту складається з кількох під етапів: перевірка прав, отримання актуального стану редактора, генерація HTML-структури, підключення CSS, формування JavaScript-анімації, обробка ресурсів і створення файлу у вибраному форматі. Для HTML-експорт важливо сформувати самодостатній документ, який може бути відкритий поза системою. Для GIF або MP4 експорт важливим є правильний початковий кадр, тривалість і відповідність часовій шкалі анімації [7; 9; 14].

Іморт HTML як новий проєкт є складнішою функцією, ніж звичайне завантаження файлу. Система повинна розпізнати структуру HTML, виділити корисні стилі, скрипти й ресурси, перетворити їх у внутрішній формат або хоча б зберегти як редагований проєкт. Через різноманітність HTML-файлів повна сумісність не гарантується, тому у документації правильно зазначати обмеження імпорт. Найкраще працює сценарій, коли імпортуються файли, раніше експортовані з цієї ж платформи [9; 14].

Одним із ризиків є надмірна концентрація логіки у великому App.jsx. На ранньому етапі прототипу це допустимо, тому що дозволяє швидко зібрати взаємодію авторизації, робочий простір, панель проєктів, редактор і платежів. Однак зі зростанням системи такий файл ускладнює підтримку. Доцільно поступово виносити логіку в сервіси, власні хуки, окремі провайдери контексту і модулі для роботи з проєктами, робочий простір, платежі, повідомлення та стан редактора [14].

Другим ризиком є залежність перевірок інтерфейсу від коду клієнтської частини. Якщо кнопка прихована для певної ролі, це покращує користувацький досвід, але не гарантує безпеку. Користувач може змінити запит або спробувати звернутися до API напряму. Тому всі важливі операції мають перевірятися на рівні RLS, RPC або Edge Functions. У пояснювальній записці це потрібно підкреслювати, оскільки така вимога демонструє інженерне розуміння різниці між зручністю інтерфейсу і реальною авторизацією [4; 10; 14].

Третім ризиком є сумісність HTML імпорт/експорт. Рекламні HTML-креативи можуть містити зовнішні бібліотеки, вбудовані скрипти, нестандартні стилі, анімації, ресурси і залежності від конкретного середовища. Якщо система повинна імпортувати будь-який HTML, складність різко зростає. Тому доцільно визначити підтримуваний підклас HTML-файлів і насамперед гарантувати стабільний цикл експорту/імпорту для файлів, створених у Creative Workflow Studio. Це робить функцію реалістичною для бакалаврського проєкту [9; 14].

У третьому розділі описано програмну архітектуру Creative Workflow Studio, визначено основні абстракції, пакети, компоненти, маршрути та сервіси.

Обґрунтовано вибір інструментів React, Vite, Supabase, PostgreSQL, GSAP, Monaco Editor та допоміжних бібліотек. Розглянуто алгоритми створення проєкту, передавання на QA-перевірку та експорту, а також показано приклади реалізації логіки клієнтської частини та синхронізації в реальному часі [1; 2; 3; 4; 5; 6; 7; 8; 11; 12; 14; 15].

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1. Тестування системи

Тестування Creative Workflow Studio має охоплювати функціональні сценарії, рольові обмеження, коректність роботи редактора, збереження даних, оновлення в реальному часі, процес попереднього перегляду та експорту, платіжний сценарій та безпеку доступів. Оскільки система має різні ролі, тестування не можна обмежувати одним користувачем: необхідно перевіряти поведінку ролей: власника, керівника, розробника, QA і гостя [14; 15].

Тестування алгоритму полягає в перевірці повного користувацького сценарію. Потрібно створити новий рекламний проєкт у панелі проєктів, перевірити появу запису в таблиці projects, відкрити редактор Studio, додати текстовий або графічний шар, зберегти зміни, передати проєкт у QA-перевірка, переконатися у створенні повідомлення для QA, відкрити попередній перегляд, повернути проєкт у стан розроблення із коментарем або перевести його у стан готовності до використання, а також перевірити експорт у HTML, GIF або MP4 [14; 15].

Окремо перевіряються права доступу: Розробник не повинен керувати чужими робочий простір без призначення, QA повинен бачити попередній перегляд і залишати коментар, Керівник або Власник мають змогу призначати відповідальних осіб і змінювати статуси робочого процесу відповідно до правил. Також перевіряється оновлення панелі проєктів у реальному часі після зміни статусу або створення нового проєкту [4; 10; 14].

У межах перевірки системи описано вимоги до програмного забезпечення з позиції власника робочого простору та кінцевого користувача, побудовано блок-схеми ключових алгоритмів, показано візуальну реалізацію сценарію, а також наведено приклади реалізації логіки на клієнтському, серверному та рівні бази даних для Creative Workflow Studio.

Тестові сценарії перевірки програмної системи наведено у табл. 4.1.

Таблиця 4.1

Тестові сценарії перевірки програмної системи

№	Тестовий сценарій і кроки виконання	Очікуваний результат
1	Створення акаунта: відкрити сторінку входу; заповнити логін і пароль; підтвердити пошту; виконати повторний вхід.	Користувач отримує сесію і переходить до первинне налаштування або сторінки проєктів.
2	Створення робочого простору: увійти в систему; перейти до первинне налаштування; вибрати персональний або командний простір; підтвердити створення.	Робочий простір з’являється у перемикачі, користувач має права власника.
3	Створення рекламного проєкту: відкрити панель проєктів; натиснути кнопку створення; ввести назву й формат; зберегти форму.	У таблиці projects створюється запис зі статусом розроблення.
4	Редагування макета: відкрити проєкт; додати текстовий або графічний шар; змінити координати, розмір, стиль; зберегти дані.	Зміни зберігаються і відновлюються після перезавантаження сторінки.
5	QA-перевірка: передати проєкт на QA; увійти як QA; відкрити попередній перегляд; погодити або повернути проєкт із коментарем.	Статус змінюється відповідно до рішення QA, а коментар доступний розробнику.
6	Експорт: відкрити погоджений проєкт; вибрати формат HTML/GIF/MP4; запустити експорт; перевірити результат.	Користувач отримує файл або матеріал для передачі в потрібному форматі.
7	Перевірка прав доступу: виконати дії від імені розробника, QA, керівника та гостя; спробувати заборонені операції.	Система дозволяє лише ті дії, що відповідають ролі користувача.
8	Оновлення в реальному часі: відкрити панель проєктів у двох сесіях; змінити статус проєкту в одній сесії.	У другій сесії список проєктів оновлюється без ручного перезавантаження.

Для тестування логіки клієнтської частини доцільно використовувати ручні сценарії та поступово додавати автоматизовані тести. На етапі бакалаврської роботи основним є функціональне тестування критичних

користувацьких сценаріїв. У подальшому варто розширити перевірку за допомогою модульних тестів для сервісів, компонентних тестів для інтерфейсу і наскрізних тестів для сценаріїв власник-розробник-QA [14; 15].

4.2. Вимоги до апаратного та програмного забезпечення

Для роботи користувача достатньо сучасного браузера, стабільного підключення до Інтернету та пристрою з екраном, придатним для роботи з редактором. Оскільки редактор Studio містить робоче полотно, панель властивостей, вікно шарів, панелі JS/CSS і часова шкала, для комфортної роботи бажано використовувати ноутбук або настільний комп’ютер. На мобільних пристроях можна переглядати окремі сторінки, однак редагування складного креативу потребує ширшого робочого простору [3; 9; 14].

Для розробника потрібні Node.js LTS, npm, Git, редактор коду, доступ до панелі керування Supabase, репозиторій клієнтська частина-застосунку, файл .env з ключами клієнтського середовища та можливість виконати підсумкова збірка. Для демонстраційного розгортання може використовуватися GitHub Pages або інший статичний хостинг, а серверна логіка при цьому залишається у Supabase [3; 4; 8; 14].

Діаграму розгортання програмної системи подано на рис. 4.1.

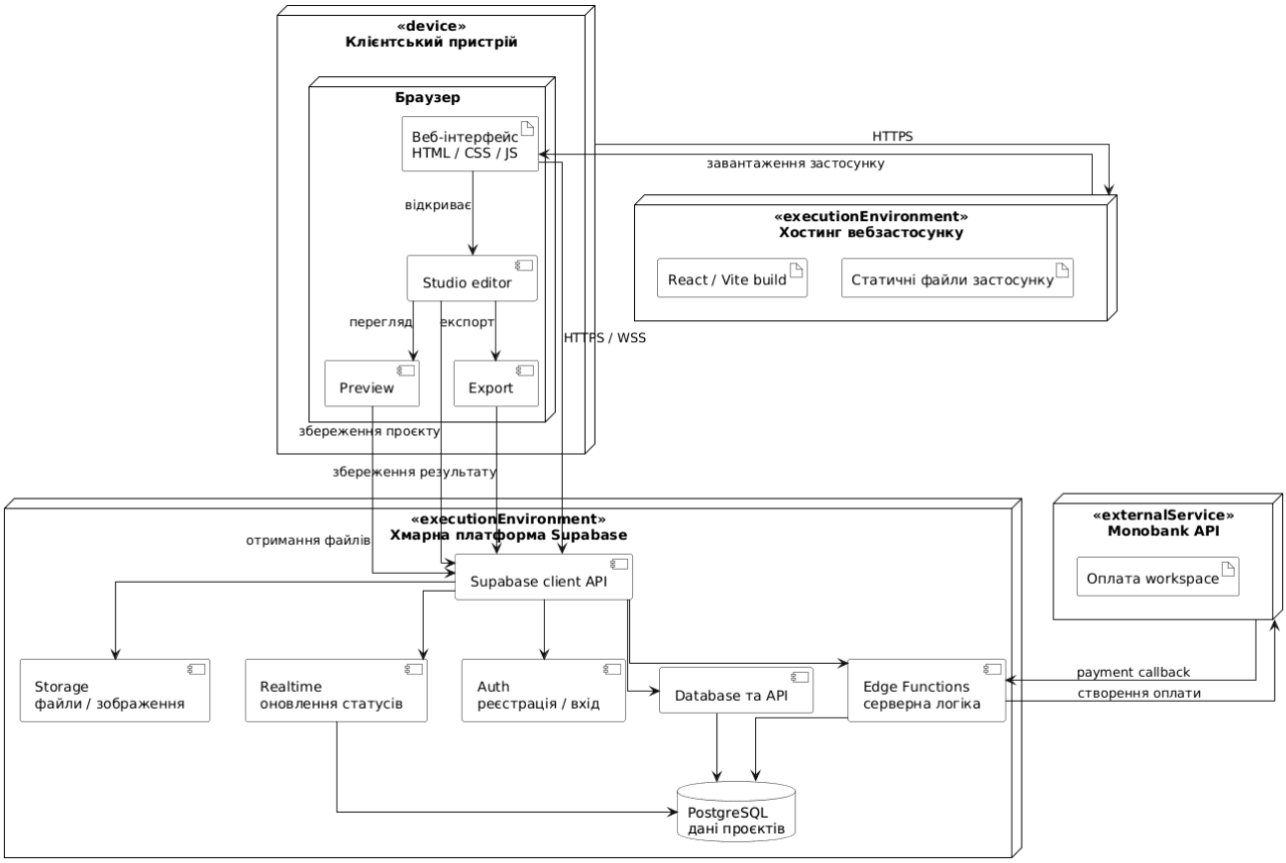

Рис. 4.1 Діаграма розгортання системи

Вимоги до апаратного та програмного забезпечення наведено у табл. 4.2.

Таблиця 4.2

Вимоги до апаратного та програмного забезпечення

Категорія	Мінімальні вимоги	Рекомендовані вимоги
Користувацький пристрій	Сучасний браузер, 4 ГБ ОЗП	Ноутбук/ПК, 8 ГБ ОЗП, стабільний Інтернет
Розробницьке середовище	Node.js LTS, npm, Git	VS Code, GitHub, локальний dev server Vite
Серверна частина-платформа	проєкт Supabase	Налаштовані Auth, PostgreSQL, RLS, Realtime, Edge Functions
Хостинг	Статичний хостинг для збірка	GitHub Pages або аналог із HTTPS
База даних	PostgreSQL у Supabase	Індекси, RLS, RPC, publication для Realtime
Браузер	Chrome/Edge/Firefox актуальної версії	Chrome або Edge для демонстрації редактора й експорту

4.3. Склад інсталяційного пакету

Склад інсталяційного пакету залежить від способу розгортання. Для клієнтської частини основним результатом є підсумкова збірка, сформований Vite. Він містить HTML, JavaScript bundles, CSS, статичні ресурси та службові файли, необхідні для роботи клієнтського застосунку. Для серверної частини інсталяційний пакет включає SQL-скрипти створення таблиць, RLS-політик, RPC-функцій, Edge Functions та інструкції з налаштування змінних середовища [3; 14].

Склад інсталяційного пакету наведено у табл. 4.3.

Таблиця 4.3

Склад інсталяційного пакету

Елемент пакету	Опис	Призначення
dist/	Збірка клієнтська частина-застосунку	Розміщення на хостингу
index.html	Точка входу вебзастосунку	Завантаження bundle і root container
assets/*.js, *.css	Згенеровані bundle-файли	Логіка застосунку та стилі
.env.example	Приклад змінних середовища	Пояснення ключів Supabase і зворотний запит URL
sql/schema.sql	DDL-структура таблиць	Створення інформаційної бази
sql/rls.sql	Політики RLS	Захист робочий простір і проєктів
supabase/functions/	Edge Functions	Платежі, зворотний запит, складні серверні операції
README.md	Інструкція з запуску	Послідовність налаштування і розгортання

Перед передачею системи до експлуатації потрібно перевірити, що секретні ключі не потрапили у клієнтська частина-збірку або репозиторій. У клієнтській частині може використовуватися лише публічний ключ, а доступ до даних має обмежуватися RLS-політиками. Секретні службові ключі мають бути

доступні лише серверним Edge Functions або захищеному середовищу [4; 8; 10; 14].

4.4. Рекомендації щодо розгортання та експлуатації

Розгортання системи доцільно виконувати поетапно. Спочатку створюється проєкт Supabase і застосовуються SQL-скрипти. Потім налаштовуються провайдери автентифікації, адреса перенаправлення, RLS-політики, публікація Realtime і Edge Functions. Після цього клієнтський застосунок збирається командою npm run build і розміщується на віддаленому хостингу. Завершальним кроком є перевірка користувацький сценарій на production URL [3; 4; 5; 6; 8; 14].

1. створити або перевірити проєкт Supabase;
2. застосувати SQL-скрипти таблиць, індексів, RLS і RPC;
3. налаштувати Auth адреса перенаправлення, підтвердження електронної пошти;
4. завантажити Edge Functions і додати необхідні змінні середовища;
5. виконати npm install і npm run build для клієнтської частини;
6. розмістити dist/ на віддаленому хостингу з HTTPS;
7. перевірити авторизацію, первинне налаштування робочого простору, створення проєкту, редактор, сценарій QA та експорт;
8. задокументувати відомі обмеження й сценарії відновлення після помилок.

У процесі експлуатації слід контролювати продуктивність панель проєктів, кількість у реальному часі-підписок, коректність RLS-політик, обсяг JSONB-даних у projects, обробку помилок платіжний зворотний запит та сумісність HTML імпорт. Для подальшого розвитку доцільно винести частину логіки з великого App.jsx, додати журналу аудиту, історію версій, шаблони рекламних форматів, детальніші права доступу і автоматизовані E2E-тести [4; 5; 10; 14].

Тестування сценарію власника має починатися з реєстрації користувача, підтвердження пошти, створення або активації робочого простору й перевірки доступу до панелі проєктів. Власник повинен бачити власний робочий простір, мати можливість створити проєкт, запросити учасника, змінити роль, переглянути список членів команди та виконати адміністративні дії. Якщо ці дії працюють некоректно, вся командна модель втрачає практичний сенс, тому сценарій власника є одним із базових для перевірки [8; 14].

Тестування сценарію розробника повинно перевіряти, чи може користувач відкрити тільки дозволені йому проєкти, змінити документ креативу, додати текстовий або графічний шар, налаштувати часову шкалу, зберегти стан і передати роботу на QA-перевірку. Також необхідно перевірити негативні сценарії: розробник не повинен мати змоги змінити ролі інших учасників, активувати чужий робочий простір або погодити статус готовності до використання, якщо така дія не передбачена політикою доступів [14; 15].

QA-сценарій має зосереджуватися на попередній перегляд, рішення перевірки і коментар. QA повинен бачити проєкти, призначені на перевірку, відкривати попередній перегляд, оцінювати результат і виконувати одну з двох дій: погодити або повернути. У разі повернення обов’язковим є текст коментар, тому що без нього розробник не отримує корисної інформації. Після виконання дії панель проєктів і повідомлення мають оновитися для інших учасників робочий простір [14; 15].

Гість-сценарій потрібний для перевірки лише для перегляду або обмеженого режиму. Якщо застосунок дозволяє режим гостя, такий користувач не повинен бачити приватні дані робочого простору, редагувати проєкти або виконувати експорт, якщо це не дозволено явно. Режим гостя може бути корисним для демонстрації інтерфейсу, але він має бути безпечним. Тому тестування гостьового режиму включає перевірку відсутності доступу до приватних таблиць і критичних дій [4; 10; 14].

Окрему групу становлять тести перезавантаження і пряме посилання. Користувач може відкрити посилання на редактор або попередній перегляд без

попереднього переходу через панель проєктів. У цьому випадку застосунок повинен відновити сесія, перевірити робочий простір, завантажити проєкт і тільки після цього показати сторінку. Якщо доступу немає, потрібно показати контрольовану помилку, а не порожній екран. Така перевірка особливо важлива для HashRouter і статичного розгортання [8; 14; 17].

Після розгортання системи необхідно регулярно перевіряти, чи відповідають налаштування Supabase фактичній структурі застосунку. Якщо додаються нові таблиці або поля, потрібно оновлювати RLS-політики, індекси, RPC-функції та типи даних у клієнтська частина. Помилки в таких налаштуваннях можуть проявлятися не відразу: наприклад, власник бачить дані, а розробник отримує порожній список через надто сувору політику або, навпаки, сторонній користувач бачить проєкти через недостатньо обмежену політику [4; 10; 14].

Для продуктивності панель проєктів важливо не завантажувати надмірні дані стан редактора, якщо на сторінці потрібні лише назва, статус, дата оновлення, призначені користувачі та короткі метадані. Якщо кожний рядок панель проєктів завантажує великий JSONB-документ з усіма шарами і кодом, це може погіршити швидкість роботи. Тому в майбутньому можна розділити коротку інформацію для списку проєктів і повні дані для редактор, або використовувати вибіркові SQL-запити [10; 14].

Realtime-синхронізація має бути налаштована обережно. З одного боку, вона підвищує зручність, тому що користувач бачить оновлення статусів і повідомлень без перезавантаження. З іншого боку, надмірна кількість підписок може створювати зайве навантаження. Доцільно підписуватися лише на актуальний робочий простір і відписуватися під час зміни робочого простору або виходу користувача. Такий підхід зменшує ризик витоку стану між різними робочими просторами [5; 14].

Для підтримки системи потрібна зрозуміла документація розгортання. Вона має описувати послідовність встановлення залежностей, створення проєкт Supabase, виконання SQL-скриптів, налаштування Auth адреса перенаправлення,

додавання змінні середовища, запуск локального серверу, формування підсумкова збірка і перевірку розгорнутого застосунку. Без такої інструкції повторне розгортання або передача проєкту іншому розробнику стає складною навіть за наявності робочого коду [3; 4; 6; 13; 14].

Особливу увагу потрібно приділити роботі з платежами. Якщо система використовує рахунок Monobank, слід враховувати час дії рахунку, можливість повторного зворотного запиту, тестову оплату і ситуації, коли користувач закрив сторінку до підтвердження платежу. Логіка серверної частини повинна бути ідемпотентною: повторне повідомлення від платіжної системи не повинно створювати дублікати або некоректно змінювати статус робочий простір. Для демонстраційної версії корисним є резервний механізм тестової оплати [14].

Рекомендований план розвитку розвитку можна поділити на короткостроковий, середньостроковий і довгостроковий. У короткостроковій перспективі доцільно покращити структуру App.jsx, винести сервіси, стабілізувати імпорт/експорт і додати більше валідацій. У середньостроковій перспективі варто реалізувати історію версій, журналу аудиту, шаблони форматів і автоматизовані тести. У довгостроковій перспективі система може отримати інтеграції з рекламними платформами, розширені ролі, спільне редагування і аналітику продуктивності креативів [14].

Розроблена система Creative Workflow Studio може вважатися готовою до демонстраційного впровадження та використання в межах навчального або тестового середовища. Вона забезпечує виконання основного циклу роботи з рекламним продуктом: від створення облікового запису й робочого простору до редагування рекламного проєкту, перевірки, погодження та експорту результату. Водночас для промислового використання система потребує подальшого розширення, зокрема поглибленого тестування, удосконалення механізмів запису дій, резервного копіювання та контролю версій.

Першим критерієм готовності є функціональна завершеність основного циклу роботи. Користувач має можливість зареєструватися, створити або обрати робочий простір, створити рекламний проєкт, відкрити його в редакторі, змінити

макет, налаштувати елементи, передати проєкт на QA-перевірку, отримати коментар або погодження, після чого виконати експорт готового рекламного продукту. Це свідчить про те, що базовий сценарій використання системи реалізовано.

Другим критерієм є коректність розмежування доступів. Система передбачає роботу з ролями користувачів і робочими просторами, що дозволяє обмежувати доступ до проєктів відповідно до прав конкретного учасника. Дані різних робочих просторів не повинні змішуватися, а користувачі мають виконувати лише ті дії, які відповідають їхній ролі в команді. Це важливо для збереження цілісності даних і контрольованості командної роботи [14; 15].

Третім критерієм є стабільність збереження та відновлення даних. Після повторного входу або перезавантаження сторінки користувач повинен бачити актуальний стан робочого простору, список рекламних проєктів, їхні статуси та збережені дані редактора. Така поведінка є необхідною умовою для практичного використання системи, оскільки робота з рекламним продуктом зазвичай виконується не за один сеанс.

Четвертим критерієм є придатність системи до демонстрації та розгортання. Програмне забезпечення може бути представлене не лише локально, а й через віддалене посилання за умови коректного збирання клієнтської частини, налаштування HTTPS-доступу, адрес перенаправлення та підключення до Supabase як серверної платформи. Це дозволяє перевірити роботу системи в умовах, наближених до реального використання [3; 14].

П’ятим критерієм є зрозумілість інтерфейсу користувача. Основні дії мають бути доступними без додаткового пояснення: створення проєкту, відкриття редактора, налаштування елементів, попередній перегляд, зміна статусу, передавання на перевірку та експорт. Зручність інтерфейсу є важливою умовою готовності системи, оскільки програмний продукт орієнтований не лише на технічних користувачів, а й на учасників команди, які працюють із рекламними матеріалами.

Шостим критерієм є документованість програмного забезпечення. Оскільки система містить клієнтську частину, серверну платформу Supabase, структуру бази даних, політики RLS, Edge Functions і сценарії роботи з проєктами, її подальша підтримка потребує опису архітектури, логіки даних, алгоритмів, обмежень і можливих напрямів розвитку. Наявність пояснювальної записки та технічної документації підвищує практичну цінність роботи й полегшує подальше доопрацювання системи [4; 6; 13; 14].

ВИСНОВКИ

У бакалаврській кваліфікаційній роботі розглянуто задачу створення веборієнтованого програмного забезпечення для створення рекламного продукту. У процесі виконання роботи було проаналізовано предметну область цифрової реклами, визначено типові проблеми виробничого процесу, пов’язані з розрізненістю інструментів, складністю контролю статусів, ручним обміном файлами, неформалізованою QA-перевіркою та відсутністю єдиного середовища для редагування, погодження й експорту.

На основі аналізу сформовано вимоги до системи Creative Workflow Studio. Вона має підтримувати створення рекламних проєктів у вебсередовищі, роботу з персональними й командними робочими просторами, рольову модель власник, керівник, розробник, QA і гість, редактор креативів, безкодового редактора часової шкали, JS/CSS редактор, попередній перегляд, експорт, коментар QA, повідомлення і оновлення в реальному часі. Така функціональність дозволяє розглядати систему як цілісну платформу робочого процесу, а не тільки як редактор.

У роботі розроблено моделі предметної області: діаграми прецедентів, послідовності, активності, класів, пакетів, компонентів, розгортання, логічну та фізичну моделі даних. Вони демонструють, як користувацькі сценарії пов’язані з інформаційними сутностями, клієнтська частина-компонентами, Supabase серверна частина і правилами доступу. Використання моделей спростило структурування програмного забезпечення і дозволило обґрунтувати поділ системи на функціональні частини.

Інформаційне забезпечення спроєктовано на основі PostgreSQL/Supabase. Основними таблицями є profiles, workspaces, workspace_members, projects, project_notifications і workspace_payments. Обґрунтовано необхідність RLS-політик, RPC і Edge Functions для захисту даних і централізованого виконання критичних операцій. Такий підхід дозволяє зберігати дані рекламних проєктів,

стан редактора, ролі, участь у робочому просторі, платежі й повідомлення в керованій реляційній структурі.

Архітектура клієнтської частини базується на React 18 і Vite. React забезпечує компонентність та керування складним станом інтерфейсу, Vite – швидку розробку й підсумкову збірку, клієнт Supabase – взаємодію з Auth, базою даних і Realtime, GSAP – професійну анімаційну логіку, Monaco Editor – зручне редагування JavaScript і CSS. Сукупність цих засобів відповідає вимогам інтерактивного вебредактораа для цифрових креативів.

Розглянуто основні алгоритми роботи системи: створення рекламного проєкту, передавання креативу на QA-перевірку, повернення з коментарем, переведення у стан готовності до використання, оновлення панелі проєктів у реальному часі та експорт у HTML/GIF/MP4. Показано, що коректна реалізація таких алгоритмів потребує узгодження клієнтських перевірок, збереження даних у PostgreSQL, створення повідомлень і серверного контролю доступів.

У роботі визначено рекомендації щодо тестування, впровадження та експлуатації системи. Запропоновано перевіряти повні користувацький сценарій для різних ролей, коректність статусів, збереження стану редактора, експорт, платіжний сценарій, доступи та синхронізацію в реальному часі. Вказано апаратні та програмні вимоги, склад інсталяційного пакету, послідовність розгортання і напрями подальшого розвитку.

Поставлена мета роботи досягнута: спроєктовано й описано веборієнтоване програмне забезпечення для створення рекламного продукту, яке поєднує редактор, робочий процес, QA-процес, рольову модель, інформаційне забезпечення та механізми розгортання. Подальший розвиток системи може бути спрямований на розширення шаблонів, журналу аудиту, історію версій, автоматизоване тестування, покращення імпорт/експорт сумісності і поглиблену інтеграцію з рекламними платформами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. React Documentation. Built-in React Hooks. URL: https://react.dev/reference/react/hooks (дата звернення: 27.04.2026).

2. React Documentation. State: A Component’s Memory. URL: https://react.dev/learn/state-a-components-memory (дата звернення: 17.05.2026).

3. Vite Documentation. Guide and build documentation. URL: https://vite.dev/guide/ (дата звернення: 05.02.2026).

4. Supabase Documentation. Row Level Security. URL: https://supabase.com/docs/guides/database/postgres/row-level-security (дата звернення: 15.04.2026).

5. Supabase Documentation. Realtime Postgres Changes. URL: https://supabase.com/docs/guides/realtime/postgres-changes (дата звернення: 17.05.2026).

6. Supabase Documentation. Edge Functions. URL: https://supabase.com/docs/guides/functions (дата звернення: 17.05.2026).

7. GSAP Documentation. Timeline. URL: https://gsap.com/docs/v3/GSAP/Timeline/ (дата звернення: 11.05.2026).

8. Supabase Documentation. Auth. URL: https://supabase.com/docs/guides/auth/ (дата звернення: 03.05.2026).

9. MDN Web Docs. Web developer guides. URL: https://developer.mozilla.org/en-US/docs/MDN/Guides (дата звернення: 10.05.2026).

10. PostgreSQL Documentation. Row Security Policies. URL: https://www.postgresql.org/docs/current/ddl-rowsecurity.html (дата звернення: 29.03.2026).

11. Monaco Editor Documentation. URL: https://microsoft.github.io/monaco-editor/ (дата звернення: 08.05.2026).

12. Recharts Documentation. URL: https://recharts.org/en-US/ (дата звернення: 30.04.2026).

13. Методичні рекомендації щодо підготовки та оформлення бакалаврської кваліфікаційної роботи для студентів освітньо-професійної програми «Інженерія програмного забезпечення». НУБіП України. Київ, 2026.

14. Самописна документація проєкту Creative Workflow Studio. Повна бізнесова, технічна та функціональна документація. Версія 0.3 Beta. Леонтович О. С., 2026.

15. Леонтович О. С. Лабораторні роботи з дисципліни «Проєктний практикум» за темою веборієнтованого програмного забезпечення для створення рекламного продукту. НУБіП України, 2026.

16. ДСТУ 8302:2015. Бібліографічне посилання. Загальні положення та правила складання. Київ: ДП «УкрНДНЦ», 2016.

17. React Router Documentation. HashRouter. URL: https://reactrouter.com/api/declarative-routers/HashRouter (дата звернення: 25.03.2026).

18. Lucide React Documentation. URL: https://lucide.dev/guide/packages/lucide-react (дата звернення: 16.04.2026).

ДОДАТОК А

Фрагменти програмної реалізації наведено у фрагментах А.1–А.7.

Фрагменти програмної реалізації

Фрагмент А.1

Програмний код реалізації сценарію

```
async function handleCreateProject(formValues) {
   if (!activeWorkspace?.id || !currentUser?.id) {
      showToast('Workspace is not available');
      return;
   }
   const name = String(formValues.name || '').trim();
   if (!name) {
      showToast('Project name is required');
      return;
   }
   const payload = {
      workspace_id: activeWorkspace.id,
      owner_id: currentUser.id,
      name,
      status: 'Development',
      format: formValues.format || '300x250',
      screen_format: formValues.orientation || 'landscape',
      developer_id: formValues.developerId || currentUser.id,
      qa_id: formValues.qaId || null,
      images: [],
      code: '',
      css_code: '',
      scene_background: '#ffffff',
      scene_border_style: 'none',
      scene_border_color: '#000000',
   };

   const createdProject = await projectService.createProject(payload);
   setProjects((prev) => [createdProject, ...prev]);

   if (formValues.openAfterCreate) {
      navigate(`/projects/${createdProject.id}/editor`);
   }
}
```

Фрагмент А.2

Програмний код реалізації сценарію

```
async function handleSendToQa(project, handoffNote) {
    if (!project?.id) {
        return;
    }

    if (!project.qa_id) {
        showToast('QA reviewer is not assigned');
        return;
    }

    await projectService.saveProject(project.id, {
        images: editorLayers,
        code: jsCode,
        css_code: cssCode,
        scene_background: sceneBackground,
        scene_border_style: sceneBorderStyle,
        scene_border_color: sceneBorderColor,
    });

    const updatedProject = await projectService.updateProjectStatus({
        projectId: project.id,
        nextStatus: 'QA Review',
        note: handoffNote || 'Creative is ready for QA review',
    });

    setProjects((prev) =>
        prev.map((item) => item.id === updatedProject.id ? updatedProject : item)
    );
    showToast('Project was sent to QA');
}
```

Фрагмент А.3

Програмний код реалізації сценарію

```
useEffect(() => {
    if (!activeWorkspace?.id) {
        return;
    }

    const channel = supabase
        .channel(`workspace-projects-${activeWorkspace.id}`)
        .on(
            'postgres_changes',
            {
                event: '*',
                schema: 'public',
                table: 'projects',
                filter: `workspace_id=eq.${activeWorkspace.id}`,
            },
            () => reloadProjects(activeWorkspace.id)
        )
        .subscribe();

    return () => {
        supabase.removeChannel(channel);
    };
}, [activeWorkspace?.id]);
```

Фрагмент А.4

Програмний код реалізації сценарію

```
serve(async (req) => {
  const authHeader = req.headers.get('Authorization') || '';
  const token = authHeader.replace('Bearer ', '');

  const supabase = createClient(
    Deno.env.get('SUPABASE_URL'),
    Deno.env.get('SUPABASE_SERVICE_ROLE_KEY')
  );

  const { data: authData } = await supabase.auth.getUser(token);
  const user = authData?.user;
  if (!user) {
    return jsonResponse({ error: 'Unauthorized' }, 401);
  }

  const { projectId, nextStatus, note } = await req.json();

  const { data: project, error: readError } = await supabase
    .from('projects')
    .select('id, workspace_id, developer_id, qa_id, status, name')
    .eq('id', projectId)
    .single();

  if (readError || !project) {
    return jsonResponse({ error: 'Project not found' }, 404);
  }

  const allowed = await canChangeProjectStatus(
    supabase,
    user.id,
    project,
    nextStatus
  );

  if (!allowed) {
    return jsonResponse({ error: 'Forbidden' }, 403);
  }

  const { data: updated, error: updateError } = await supabase
    .from('projects')
    .update({
      status: nextStatus,
```

```
        qa_handoff_note: note || null,
        updated_at: new Date().toISOString(),
      })
      .eq('id', projectId)
      .select('*')
      .single();

  if (updateError) {
    return jsonResponse({ error: updateError.message }, 400);
  }

  await supabase.from('project_notifications').insert({
    workspace_id: project.workspace_id,
    project_id: project.id,
    recipient_id: project.qa_id,
    actor_id: user.id,
    type: 'project_status_changed',
    title: `${project.name} moved to ${nextStatus}`,
    body: note || null,
    is_read: false,
    created_at: new Date().toISOString(),
  });

  return jsonResponse({ project: updated }, 200);
});
```

Фрагмент А.5

Програмний код реалізації сценарію

```
export async function createProject(payload) {
    const { data, error } = await supabase
        .from('projects')
        .insert({
            workspace_id: payload.workspace_id,
            owner_id: payload.owner_id,
            name: payload.name,
            status: 'Development',
            format: payload.format,
            screen_format: payload.screen_format,
            developer_id: payload.developer_id,
            qa_id: payload.qa_id,
            images: payload.images || [],
            code: payload.code || '',
            css_code: payload.css_code || '',
            scene_background: payload.scene_background || '#ffffff',
            scene_border_style: payload.scene_border_style || 'none',
            scene_border_color: payload.scene_border_color || '#000000',
            created_at: new Date().toISOString(),
            updated_at: new Date().toISOString(),
        })
        .select('*')
        .single();

    if (error) {
        throw error;
    }

    return data;
}
```

Фрагмент А.6

Програмний код реалізації сценарію

```
export async function saveProject(projectId, editorState) {
    const { data, error } = await supabase
        .from('projects')
        .update({
            images: editorState.images,
            code: editorState.code,
            css_code: editorState.css_code,
            scene_background: editorState.scene_background,
            scene_border_style: editorState.scene_border_style,
            scene_border_color: editorState.scene_border_color,
            updated_at: new Date().toISOString(),
        })
        .eq('id', projectId)
        .select('*')
        .single();

    if (error) {
        throw error;
    }

    return data;
}
```

Фрагмент А.7

Програмний код реалізації сценарію

```
async function returnProjectToDevelopment(project, feedbackText, actorId) {
   const { data: updatedProject, error } = await supabase
      .from('projects')
      .update({
         status: 'Development',
         qa_feedback_note: feedbackText,
         updated_at: new Date().toISOString(),
      })
      .eq('id', project.id)
      .select('*')
      .single();

   if (error) {
      throw error;
   }

   await supabase.from('project_notifications').insert({
      workspace_id: project.workspace_id,
      project_id: project.id,
      recipient_id: project.developer_id,
      actor_id: actorId,
      type: 'project_returned_to_development',
      title: `${project.name} returned to Development`,
      body: feedbackText,
      is_read: false,
      created_at: new Date().toISOString(),
   });

   return updatedProject;
}
```

ДОДАТОК Б

Додаткові діаграми предметної області, структури даних, пакетів і компонентів подано на рис. Б.1–Б.7.

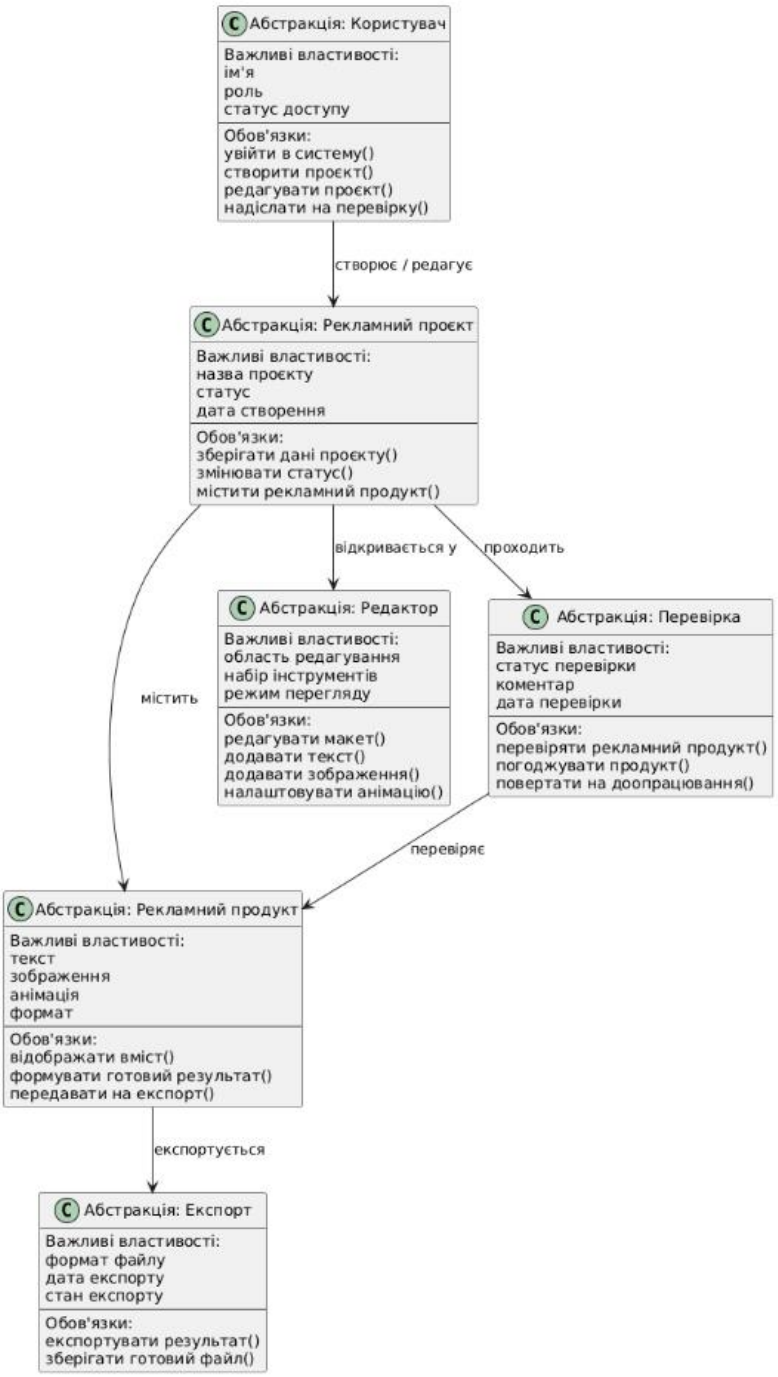

Рис. Б.1 Діаграма абстракцій предметної області

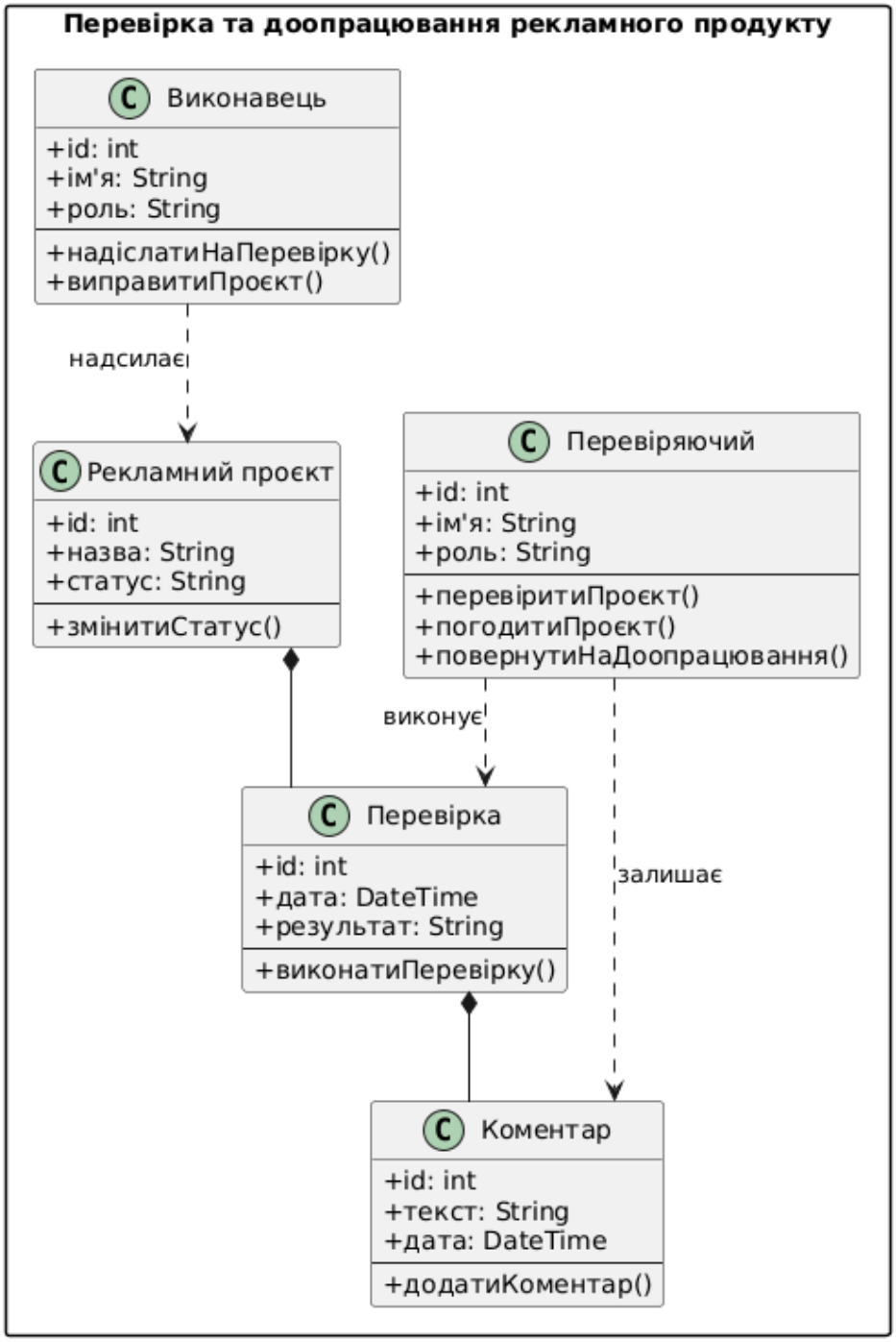

Рис. Б.2 Додаткова ER-діаграма логічної структури

Рис. Б.3 Деталізація логічної моделі даних

Рис. Б.4 Діаграма потоків робочий процес

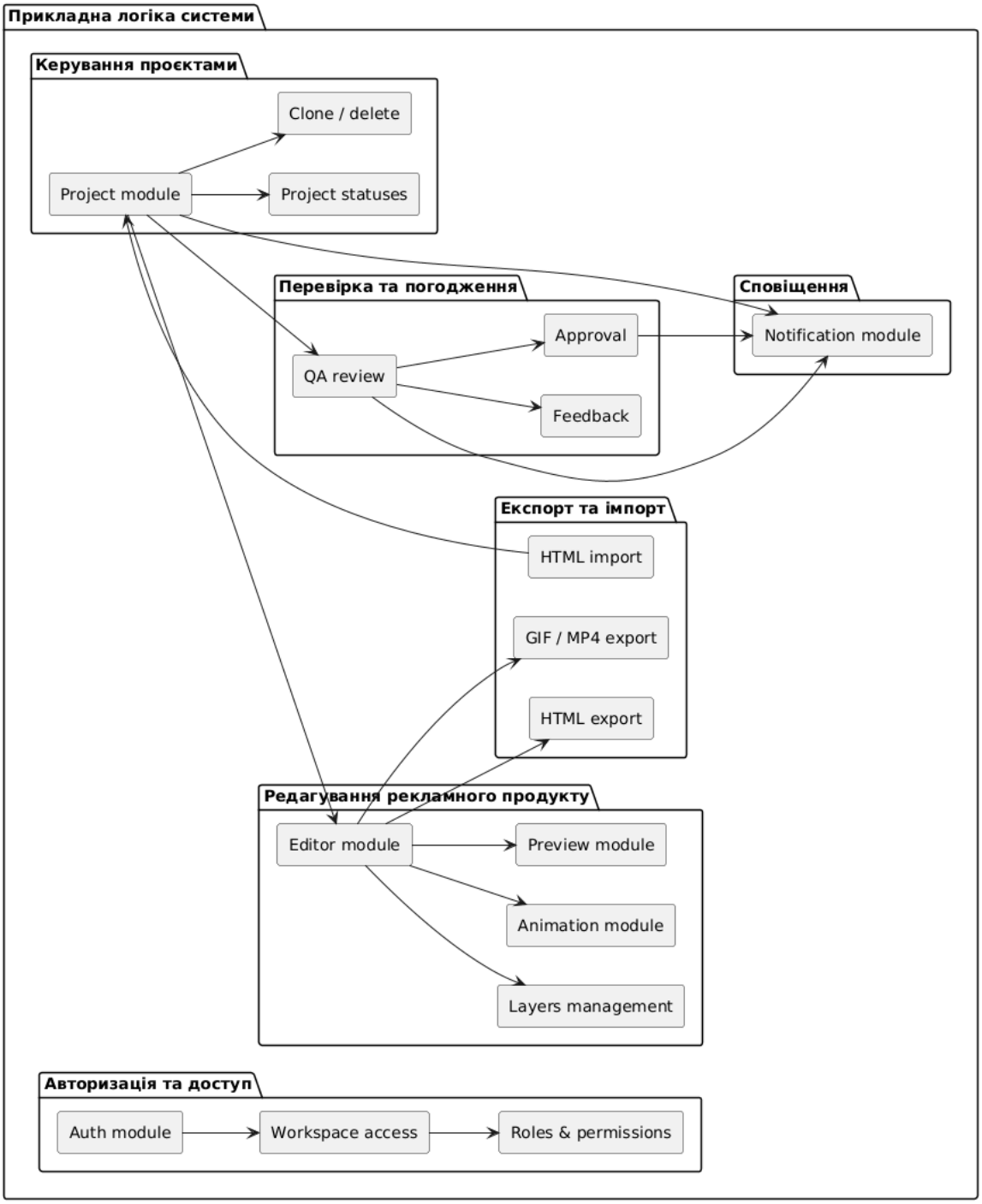

Рис. Б.5 Деталізація пакетної структури

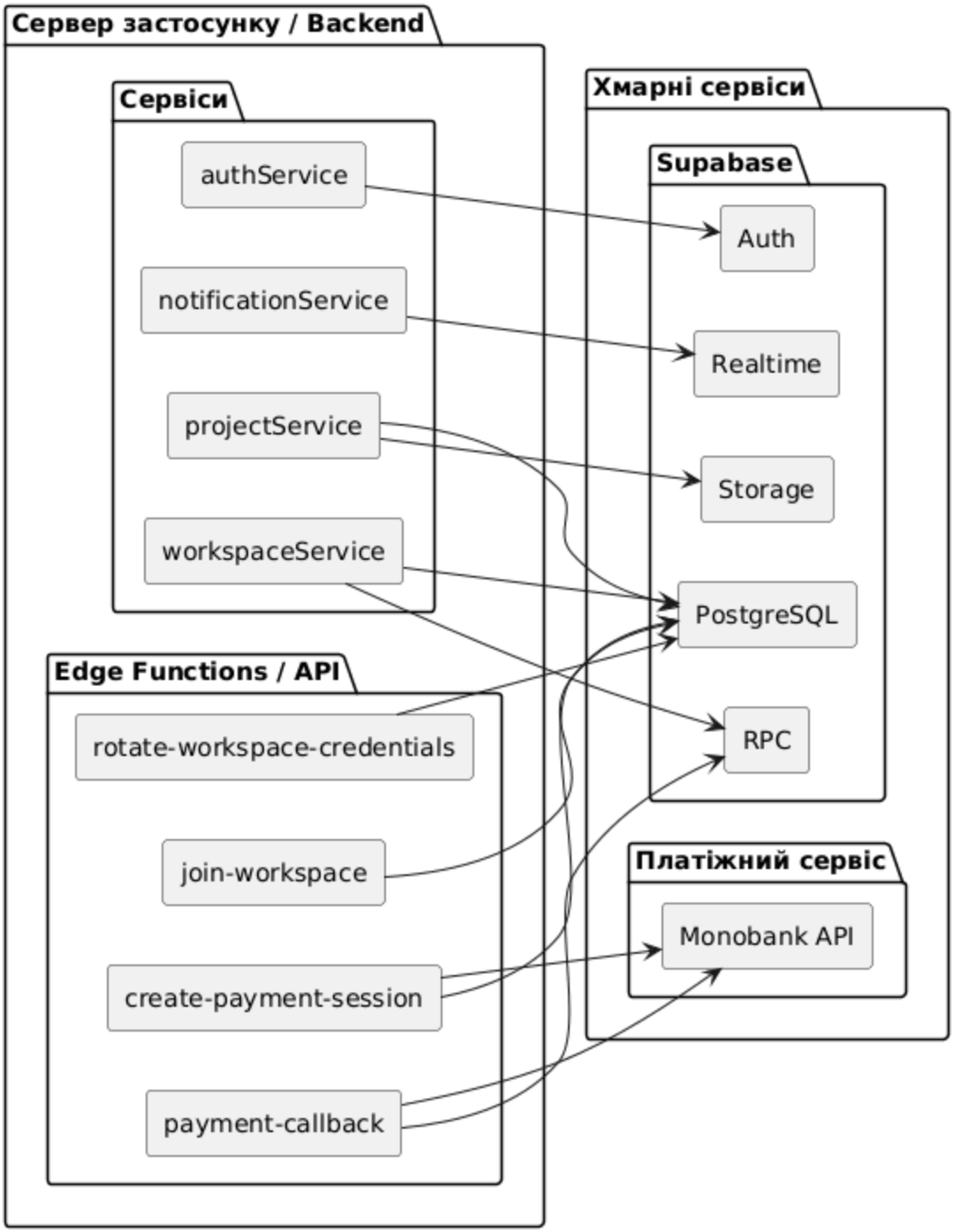

Рис. Б.6 Діаграма взаємодії компонентів

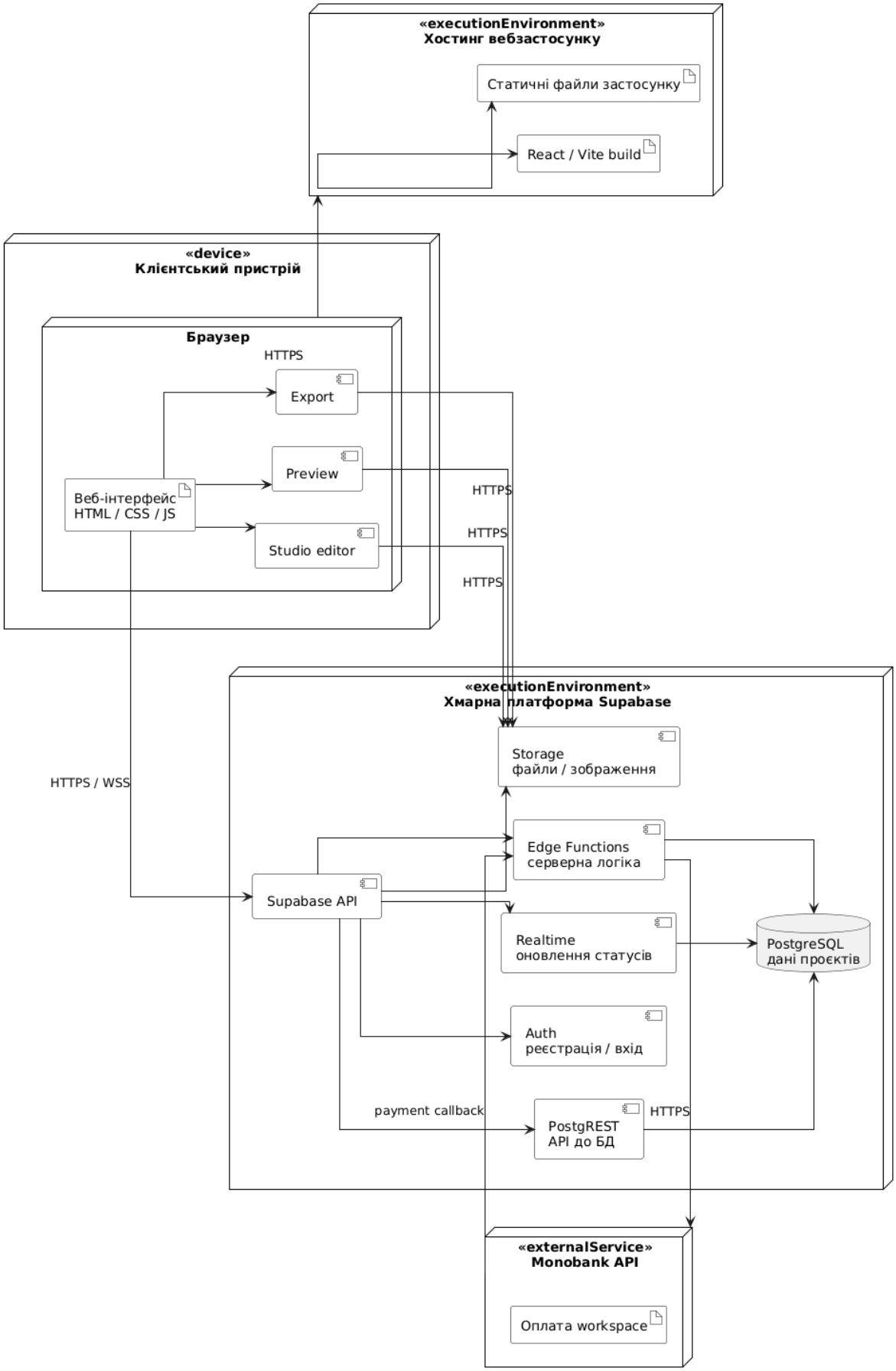

Рис. Б.7 Комплексна діаграма компонентів, артефактів і вузлів

ДОДАТОК В

Скриншоти інтерфейсу програмного забезпечення наведено на рис. В.1–В.11.

Скриншоти користувацького інтерфейсу

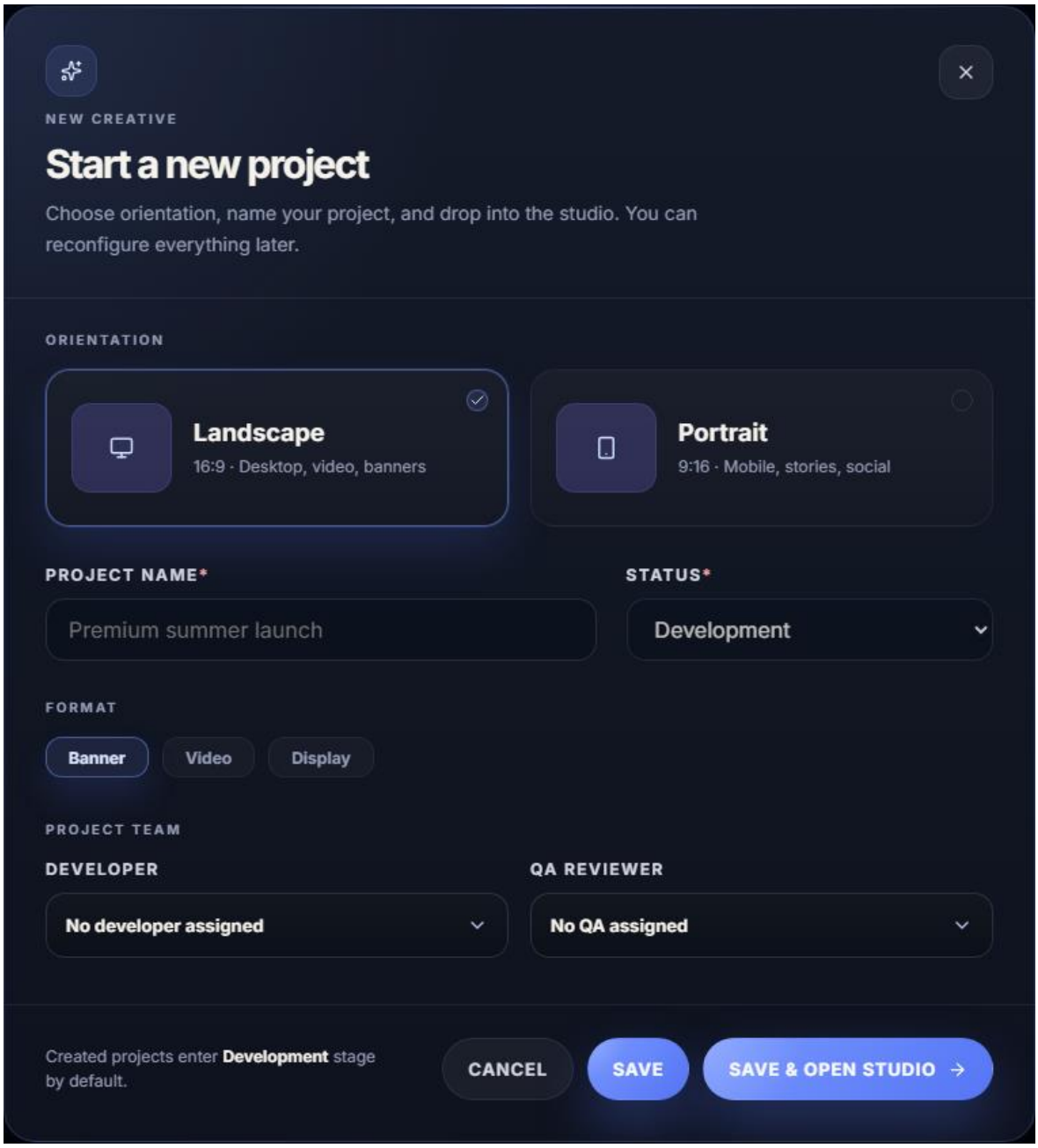

Рис. В.1 Модальне вікно створення нового рекламного проєкту

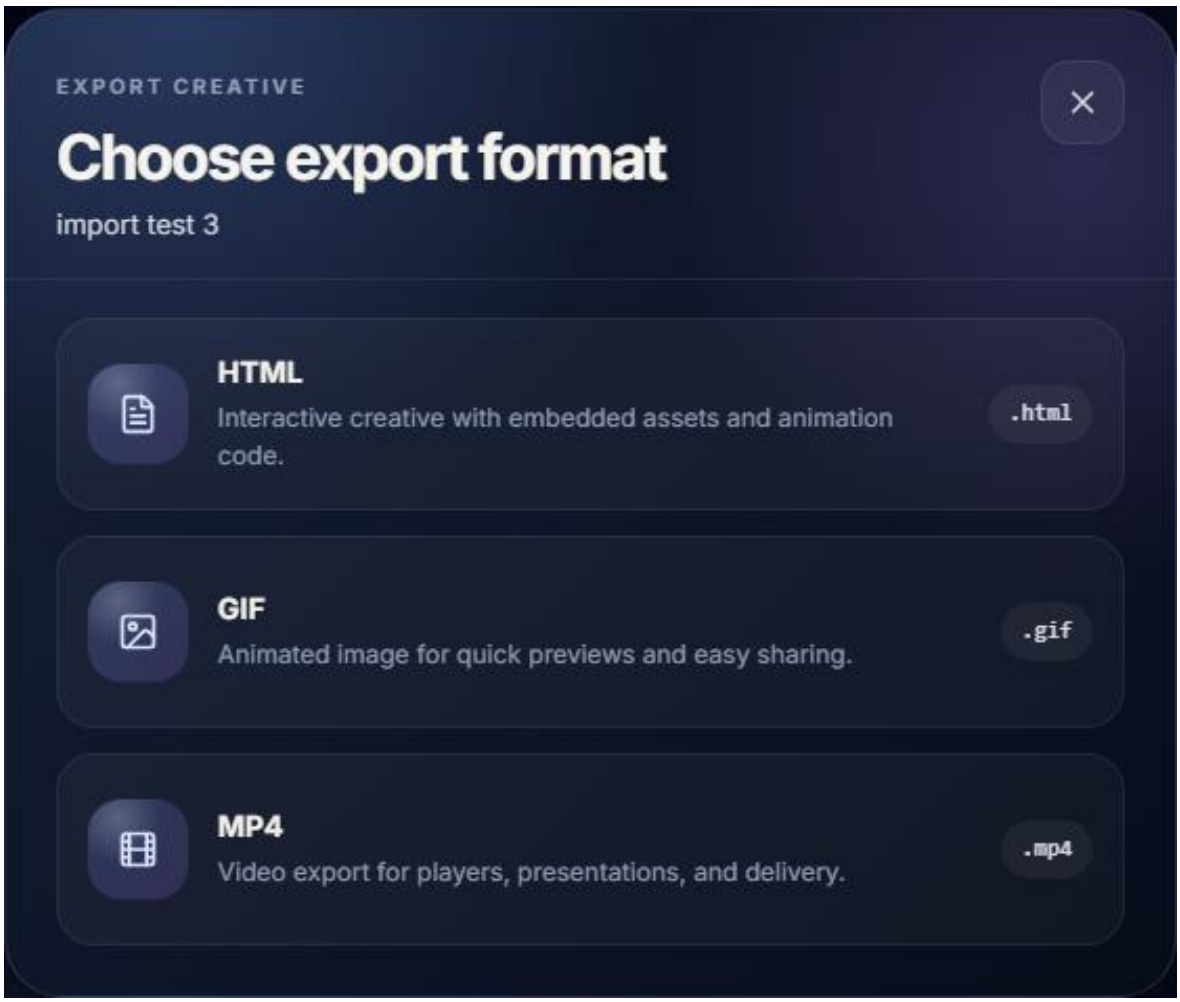

Рис. В.2 Вікно вибору формату експорту

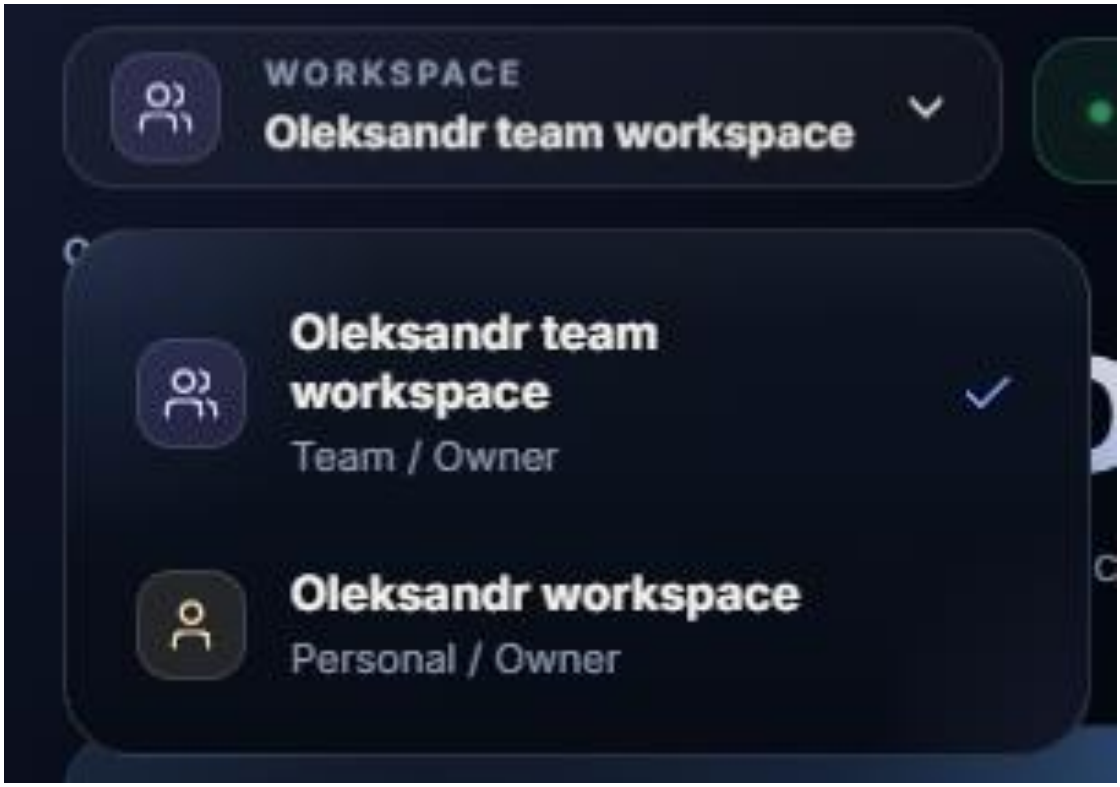

Рис. В.3 Перемикач робочий простір

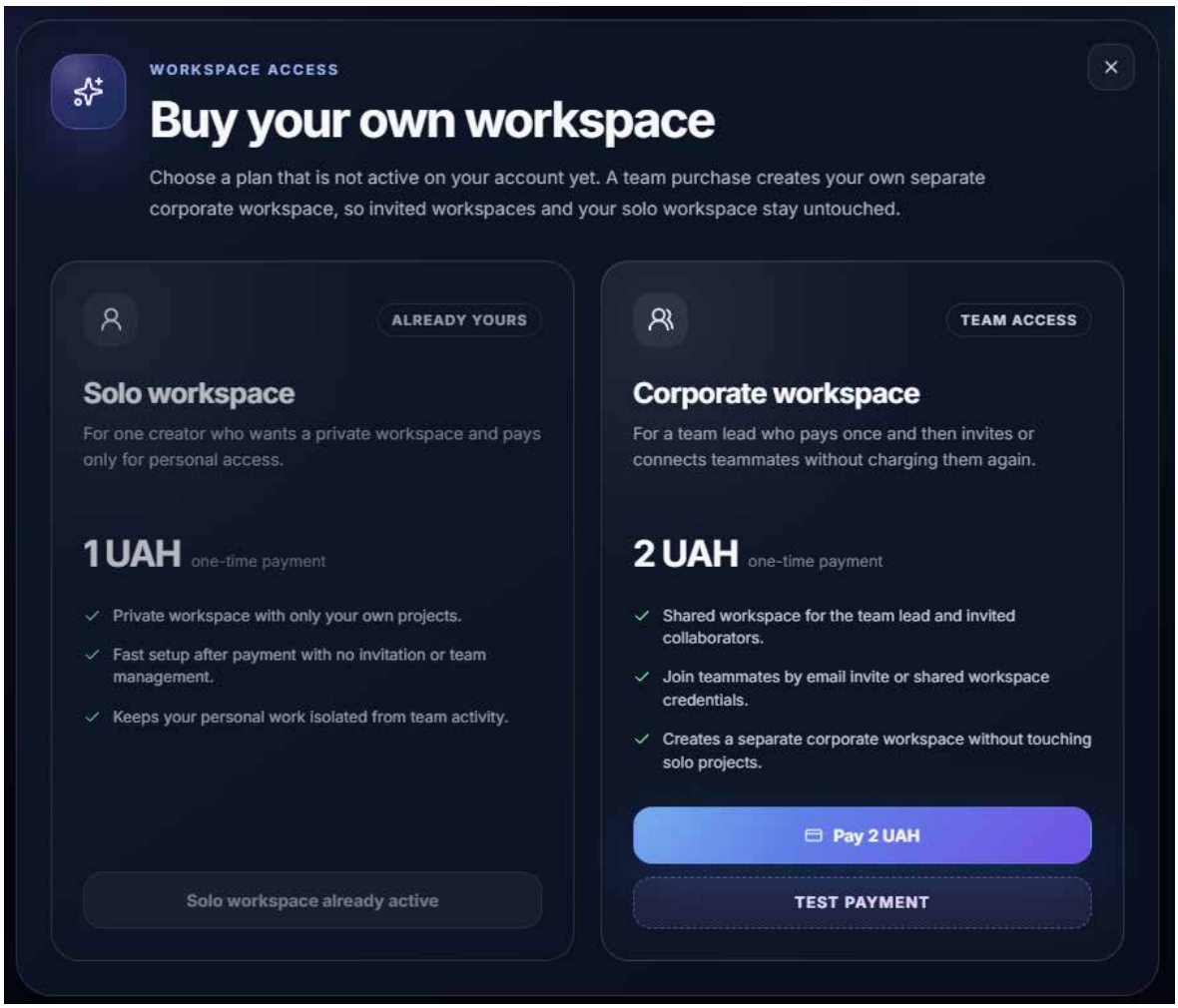

Рис. В.4 Модальне вікно покупки робочий простір

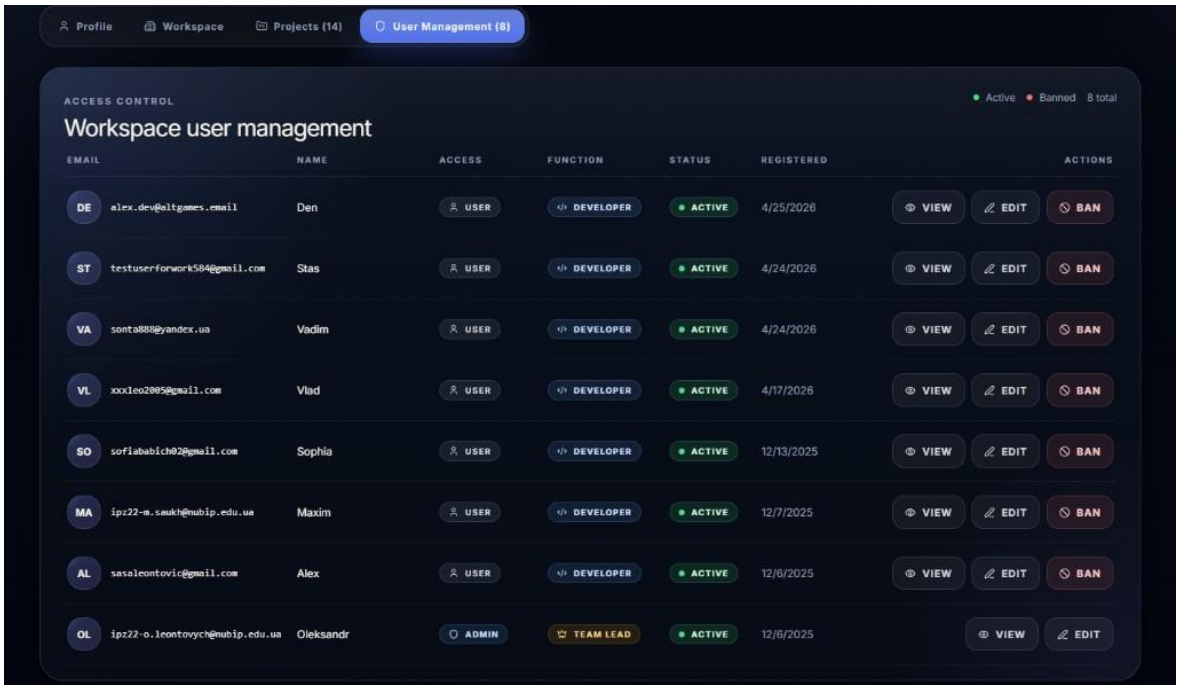

Рис. В.5 Менеджмент користувачів у робочому просторі

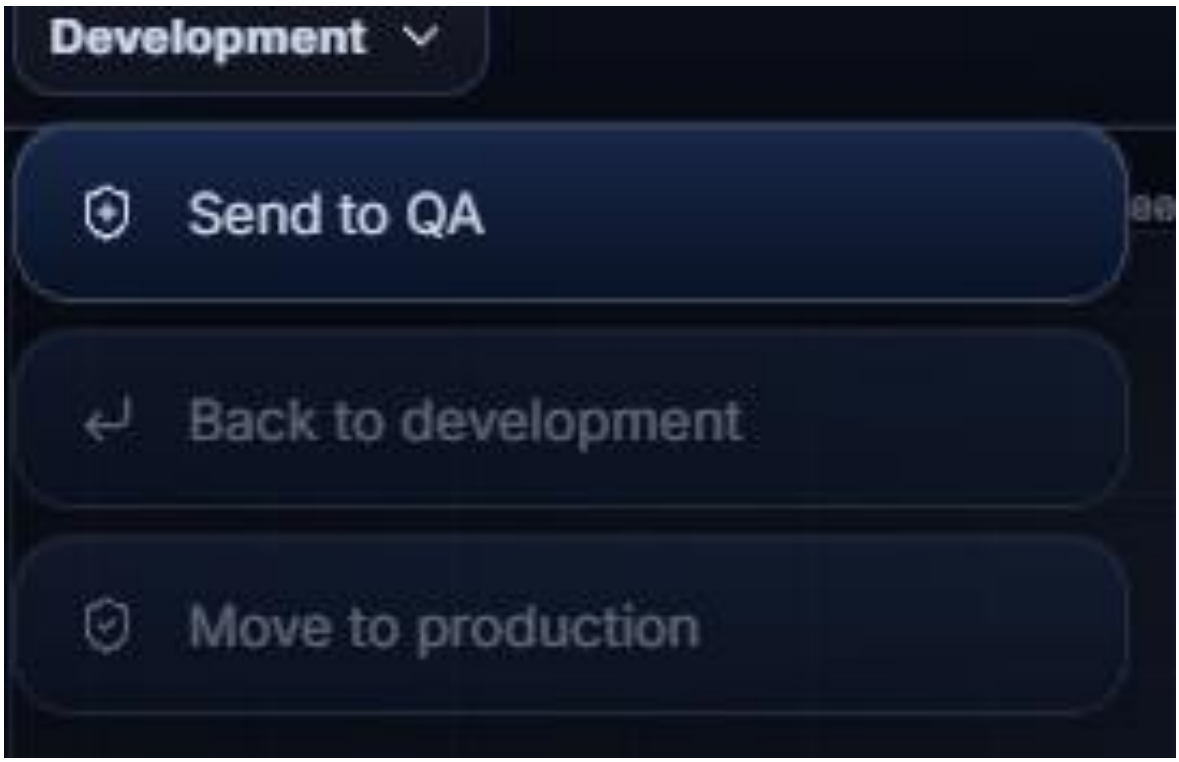

Рис. В.6 Меню статусу робочого процесу

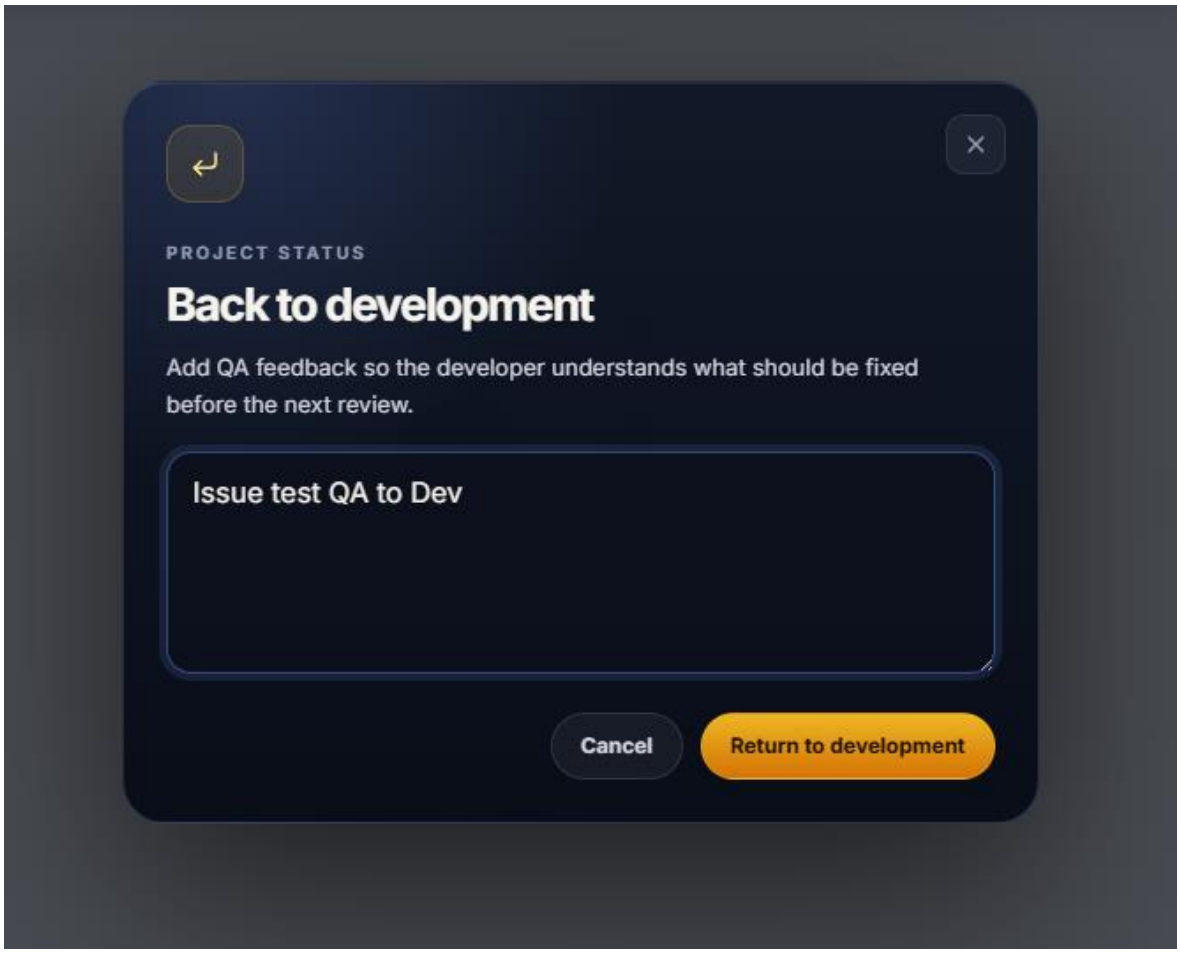

Рис. В.7 Повернення проєкту з коментар QA

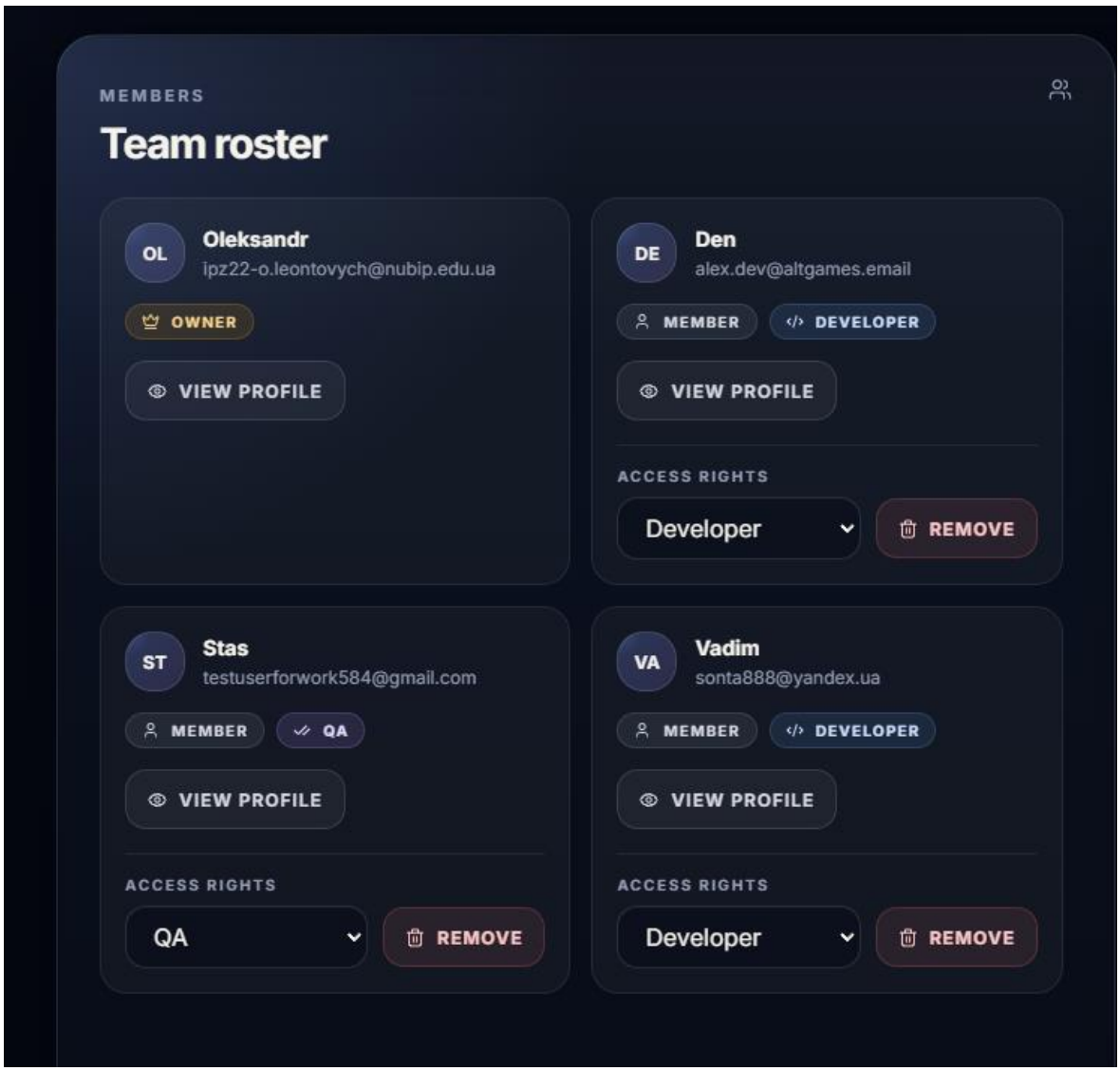

Рис. В.8 Team roster і рольові права

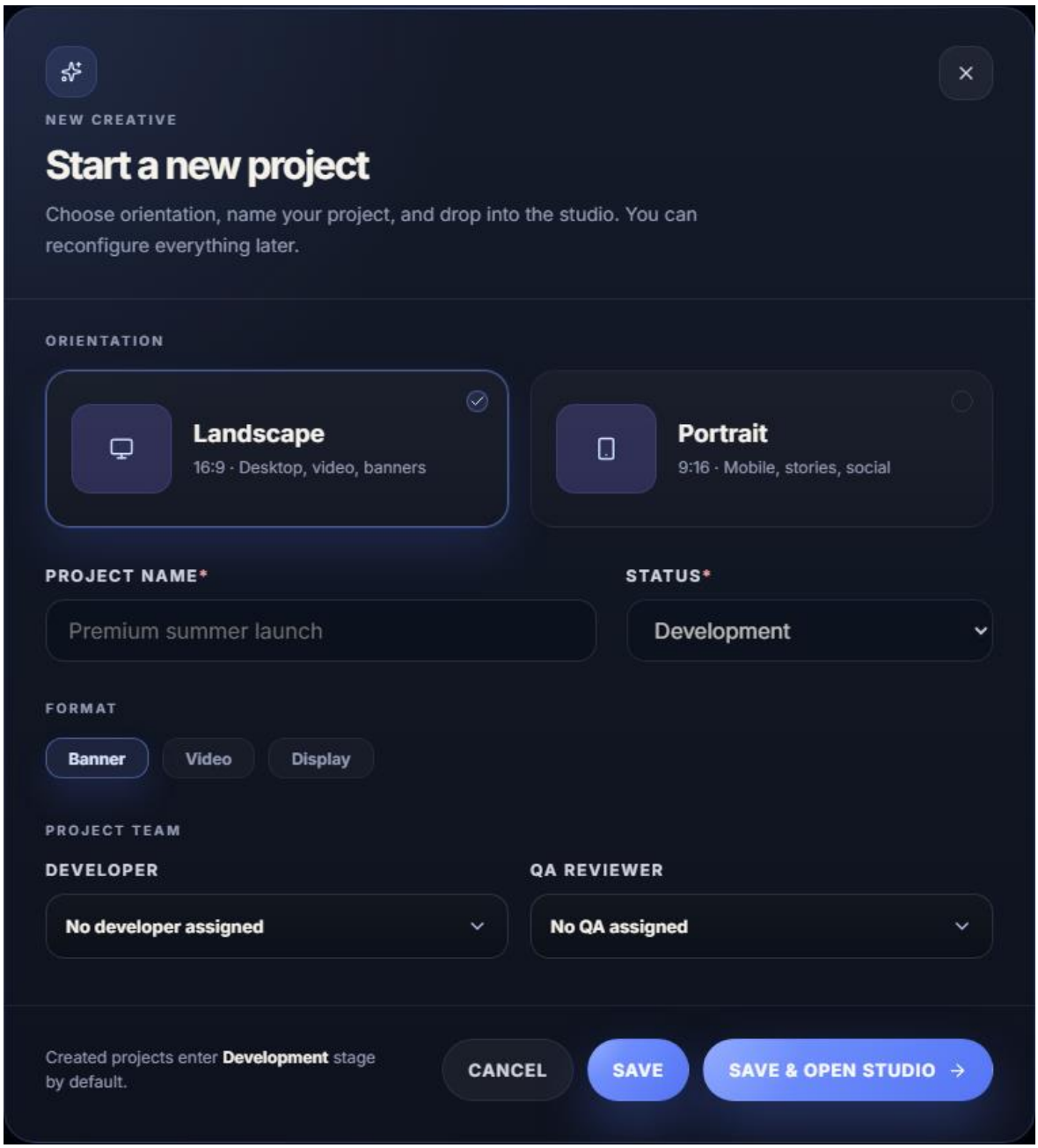

Рис. В.9 Форма створення нового проєкту

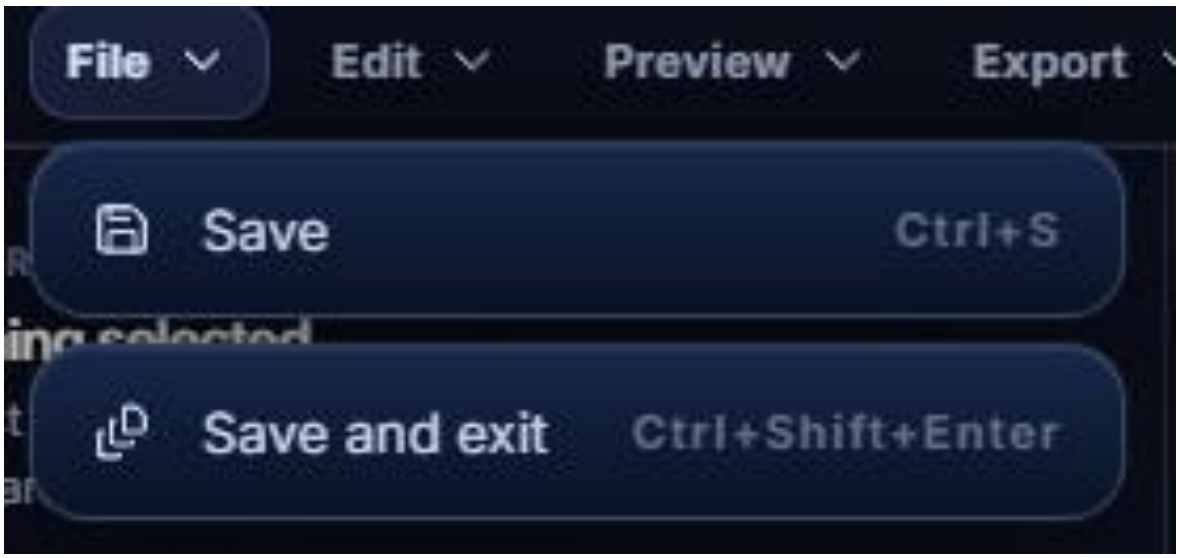

Рис. В.10 Команда збереження та виходу

Рис. В.11 Меню редагування JS/CSS/часова шкала

ДОДАТОК Г

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій
Декларація про академічну доброчесність та використання ШІ

Я, Леонтович Олександр Сергійович, здобувач(ка) НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Веборієнтоване програмне забезпечення для створення рекламного продукту» 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):

o [ ] інструменти генеративного ШІ не використовувалися.

o **[+]** інструменти ШІ ChatGPT та Claude

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. _____ **Олександр Леонтович**

(підпис)