Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавра

з напряму підготовки 121 «Інженерія програмного забезпечення»

На тему: Розробка вебплатформи арткураторства

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.

Студентка гр. ІПЗ-21-2

_____ В. В. Машкіна

Керівник кваліфікаційної роботи _____ / О. Г. Рибальченко /

Завідувач кафедри _____ / А. М. Стрюк /

Кривий Ріг
2025

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Ступінь вищої освіти: бакалавр

Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри

_____ А. М. Стрюк

«___» _____ 20__ р.

ЗАВДАННЯ

на кваліфікаційну роботу

студентці групи ІПЗ-21-2 Машкіній Вікторії Володимирівні

1. Тема: Розробка вебплатформи арткураторства затверджено наказом по КНУ № 200с від «10» квітня 2025 р.
2. Термін подання студентом закінченої роботи: «12» червня 2025р.
3. Вихідні дані по роботі: митець/киня, артворк, куратор.
4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): сегментувати таргет; сформулювати кураторську модель; зосередити увагу на створенні візуальної складової; налаштувати комунікацію між митцями/кинями та кураторами; транслювати бачення митця/кині у повній мірі; відповідати сучасним практикам цифрового мистецтва.
5. Перелік ілюстративного матеріалу: конструкції візуальної складової, діаграма сегментів, діаграми послідовності, композиція ґрід-системи, скріншоти базових UI-макетів та фінального UI вебплатформи.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Огляд літератури	15.04.2025 – 18.04.2025
2	Аналіз референсів сучасних цифрових платформ	19.04.2025 – 24.04.2025
4	Формулювання вимог та розробка концептуальної моделі	25.04.2025 – 27.04.2025
5	Підготовка матеріалів першого розділу роботи	28.04.2025 – 30.04.2025
6	Створення візуальної складової, композиції ґрід-системи та базових UI-макетів вебплатформи	01.05.2025 – 05.05.2025
7	Підготовка матеріалів другого розділу роботи	06.05.2025 – 08.05.2025
8	Вибір технологій та проєктування бази даних	09.05.2025 – 11.05.2025
9	Реалізація фронтенду вебплатформи	12.05.2025 – 20.05.2025
10	Реалізація бекенду вебплатформи	21.05.2025 – 31.05.2025
11	Тестування	01.06.2025 – 03.06.2025
12	Підготовка матеріалів третього розділу роботи	04.06.2025 – 06.06.2025
13	Оформлення пояснювальної записки	07.06.2025 – 10.06.2025

Дата видачі завдання «12» квітня 2025 р.

Студентка _____ / В. В. Машкіна /

Керівник роботи _____ / О. Г. Рибальченко /

РЕФЕРАТ

ВЕБПЛАТФОРМА, ЦИФРОВЕ МИСТЕЦТВО, КРЕАТИВНЕ КОДУВАННЯ, ВІЗУАЛЬНЕ ПРОГРАМУВАННЯ, МИТЕЦЬ, КУРАТОР, КУРАТОРСТВО, ОПЕНКОЛ, АРТВОРК

Пояснювальна записка: 51 с., 1 табл., 55 рис., 1 дод., 15 джерел.

Метою кваліфікаційної роботи є створення платформи, яка одночасно експонує та архівує сучасне українське цифрове мистецтво.

У теоретичному блоці виконано контекстуальний та концептуальний огляди: сформульовано визначення сучасного цифрового мистецтва, здійснено порівняльний аналіз референсів, визначено цільову авдиторію, створено візуальну складову проєкту, спроєктовано модель опенколу та композицію ґрід-системи, а також підготовлено базові UI-макети вебплатформи.

У практичному блоці повністю реалізовано вебплатформу арткураторства, представлено фінальний інтерфейс та проведено тестування.

Вебплатформа орієнтована на митців/кинь, внутрішніх та зовнішніх кураторів, зацікавлену авдиторію; у довгостроковій перспективі слугує архівним репозиторієм українських експериментальних цифрових форматів та може інтегруватися зі схожими ініціативами.

ABSTRACT

WEB PLATFORM, DIGITAL ART, CREATIVE CODING, VISUAL PROGRAMMING, ARTIST, CURATOR, CURATION, OPEN CALL, ARTWORK

Thesis in 51 p., 1 tab., 55 fig., 1 app., 15 references.

The goal of the thesis is to create a platform that simultaneously exhibits and archives contemporary Ukrainian digital art.

The theoretical section presents contextual and conceptual surveys: it formulates a definition of contemporary digital art, conducts a comparative analysis of reference projects, identifies the target audience, develops the project's visual language, designs the open call model and the grid-system layout, and prepares the basic UI mock-ups of the web platform.

The practical section contains the full implementation of the art-curation web platform, showcases the final interface, and reports on the testing carried out.

The platform is intended for artists, in-house and external curators, and an interested audience; in the long term it will serve as an archival repository of Ukrainian experimental digital formats and can be integrated with similar initiatives.

ЗМІСТ

ВСТУП

Мистецтво за своєю суттю веде постійний діалог з минулим: митці/кині реінтерпретують та трансформують минулі культурні форми самовираження, наративи та техніки, щоб говорити новими контекстами. Такий діалог являє собою ітеративний процес, що стає водночас як основою, так і результатом для мистецьких пошуків в секторі сучасного мистецтва. Цифрове мистецтво становить нинішню фазу історичного самопереосмислення мистецтва, що відходить від традиційних форм до інтерактивних, технологічно насичених форматів.

Втім, сучасне українське цифрове мистецтво стикається зі значними викликами. Попри свою інноваційність, воно залишається фрагментарним, а його цільова авдиторія розрізнена між окремими акаунтами в соціальних мережах та короткотривалими ініціативами. До того ж, виникають ускладнення через воєнний конфлікт та породжену ним нестабільність, внаслідок чого фізичні виставкові простори стали недоступними або суттєво обмеженими.

З огляду на вищевиявлені виклики, проблематика, актуальність та мета кваліфікаційної роботи сходяться до однієї точки: створення платформи, яка одночасно експонує та архівує сучасне українське цифрове мистецтво.

Цілями такої платформи будуть:

1. Визначити та сегментувати таргет;
2. Сформулювати прозору кураторську модель із зазначенням критеріїв відбору;
3. Зосередити увагу на створенні візуальної складової проєкту;
4. Налаштувати комунікацію між митцями/кинями та кураторами платформи;
5. Транслювати бачення митця/кині у повній мірі, з мінімальним втручанням з боку кураторів платформи;
6. Відповідати сучасним практикам цифрового мистецтва.

1 КОНТЕКСТУАЛЬНИЙ ОГЛЯД

1.1 Визначення сучасного цифрового мистецтва

Багато митців/кинь, що працюють в межах сучасного цифрового мистецтва, розглядають комп’ютерні технології як скульптурну «матерію», створюючи артворки, які часто існують як інтерактивні системи, а не як фіксовані, нерухомі об'єкти. Їхній виклик – зробити нематеріальне настільки ж відчутним та провокативним, як і щось матеріальне; їхня практика – відшукати та відтворити через дослідження та експериментування нові форми, яких до того просто не існувало.

$$\text{Матеріал} = \text{код} + \text{параметри} + \text{real-time дані}, \tag{1.1}$$

де *«глина»* – це чиста логіка; артворк є живим лише доти, доки його компілює комп'ютер.

$$\text{Об'єкт} \rightarrow \text{система}, \tag{1.2}$$

де глядач не просто *дивиться* на артворк, а *взаємодіє* з ним; артворк ніколи не буває однаковим двічі.

Сучасне цифрове мистецтво керується чотирма взаємопов'язаними силами – видовищністю, адаптивністю, залученістю спільноти та монетизацією. Дві практики, які часто досягають між ними «золотої середини» та постають як ключові методи роботи митця/кині з цифровим матеріалом, – креативне кодування та візуальне програмування. Розділення між ними радше умовне, зумовлене переважно різним інструментарієм та бекґраундом митця/кині, але, зрештою, обидві практики дозволяють реалізувати ідеї через втручання технологій – алгоритми.

Креативне кодування – це text-based різновид програмування, основним наміром якого є насамперед дослідження, гра або розповідь замість

вирішення суто технічної задачі. Така практика оперує текстом – так званим «скетчем»: ідея швидкого написання коду, ітерацій, а потім його виконання, внаслідок яких і «стається» артворк, є центральною для цього напряму. Екран виконує роль полотна; «фарбою» слугують потоки даних та математичні трансформації в реальному часі. Так, митці/кині здебільшого «малюють» завдяки таким фреймворкам, як p5.js, Processing і/або ін.

Візуальне програмування – альтернативний node/patch-based підхід до програмування. Така практика оперує наочною композицією, завдяки якій технологічний бар'єр суттєво знижується: митець/киня фокусується на тому, які саме ноди додати до композиції та як саме їх поєднати між собою. Популярними прикладами таких нодових візуальних інструментів є TouchDesigner, vvvv і/або ін.

1.2 Порівняльний аналіз референсів

Аналіз референсів формує конкретну відправну точку для проєктних рішень: порівняльні дані викривають не тільки патерни взаємодії користувачів та їх очікування, а й прогалини, які проєктована платформа повинна усунути, аби виправдати своє існування. Іншими словами, отримані цінні інсайти у тривалій перспективі дозволять суттєво знизити ризики, пов'язані з використанням неактуальних, непрактичних або неперевірених концепцій.

Серед референсів, які необхідно врахувати в контексті порівняння, можна виокремити Artsy [1] – глобальну платформу для колекціонування та відкриття мистецтва: платформа поєднує між собою митців/кинь, галереї, аукціони, колекціонерів та, власне, саму публіку. Зосереджується на функціях, що полегшують процес придбання артворків. Мета – розширювати артринок та підтримувати сталий розвиток творчих практик (Рисунок 1.1).

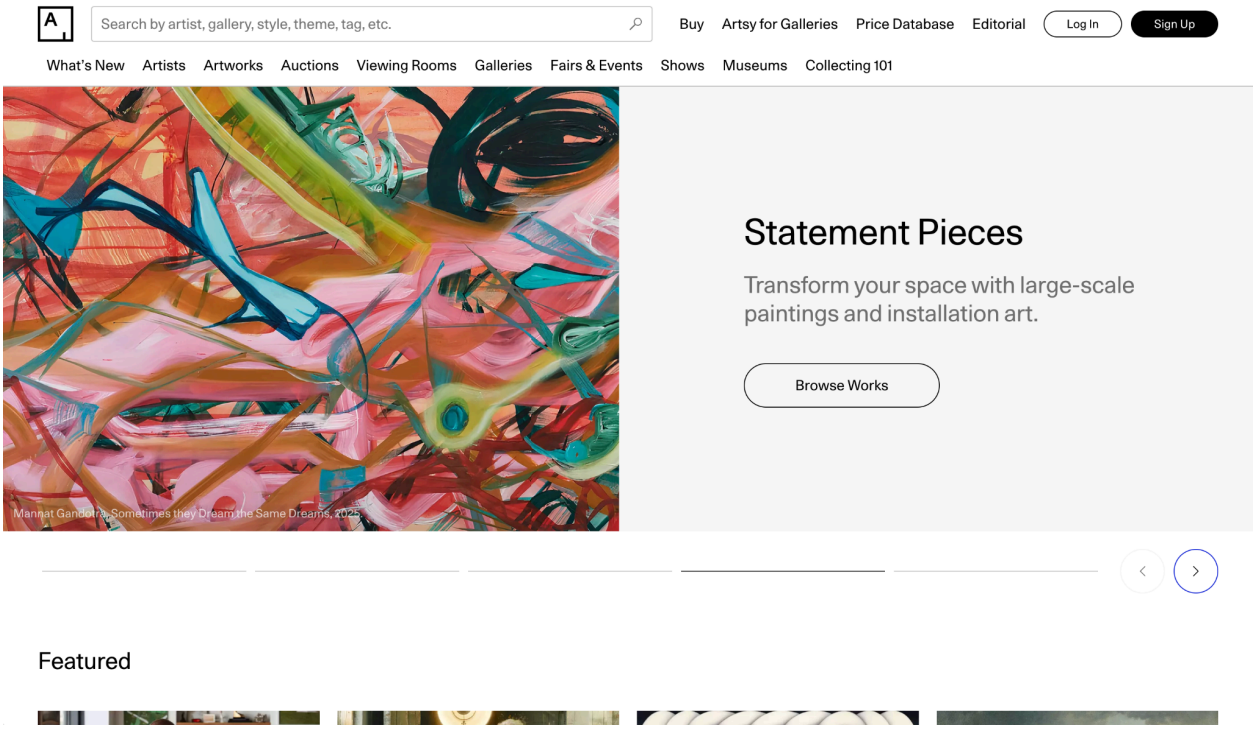

Рисунок 1.1 – Головна сторінка вебплатформи «Artsy»

Сильні сторони Artsy:

1. Розширені можливості пошуку включають багаторівневу фільтрацію за категоріями What's New/Artists/Artworks/та ін.;
2. Ті артворки, що сподобались користувачу, збираються в Saves; опубліковані користувачем артворки – в Collections (в такому випадку користувач розпізнається як митець/киня);
3. View in Room – AR-розміщення для артворків, які користувач має можливість переглядати буквально в поточному для нього просторі;
4. Алгоритмічна персоналізація дозволяє Artsy пропонувати більш релевантні та цільові рекомендації – New Works for You – на основі попередніх переглядів/Saved Artworks користувача;
5. Чиста композиція з достатньою кількістю білого простору;
6. Фокус на якісних big-sized медіа, які спрямовують увагу користувача на тонкі нюанси і/або текстуру артворку;
7. Стримана палітра кольорів;

8. Респонсивний дизайн.

Слабкі сторони Artsy:

1. Залежність від партнерства: переважна частина артворків висвітлюється завдяки співпраці з галереями, музеями та аукціонами. Зміна і/або розрив цих партнерств, ймовірно, можуть зменшити обсяг представлених артворків на платформі;
2. Надмірна комерціалізація. Сильний акцент на продаж може негативно впливати на сприйняття авдиторії, яка зацікавлена більше в дослідженні, а не в здійсненні комерційних операцій.

Spline Art [2] – платформа, що вже понад п’ять років просуває сучасне українське мистецтво й вибудовує міст між визнаними та молодими художниками, колекціонерами й усіма, хто цінує творчість. Мета – зробити український артсектор доступним та глобально інтегрованим (Рисунок 1.2).

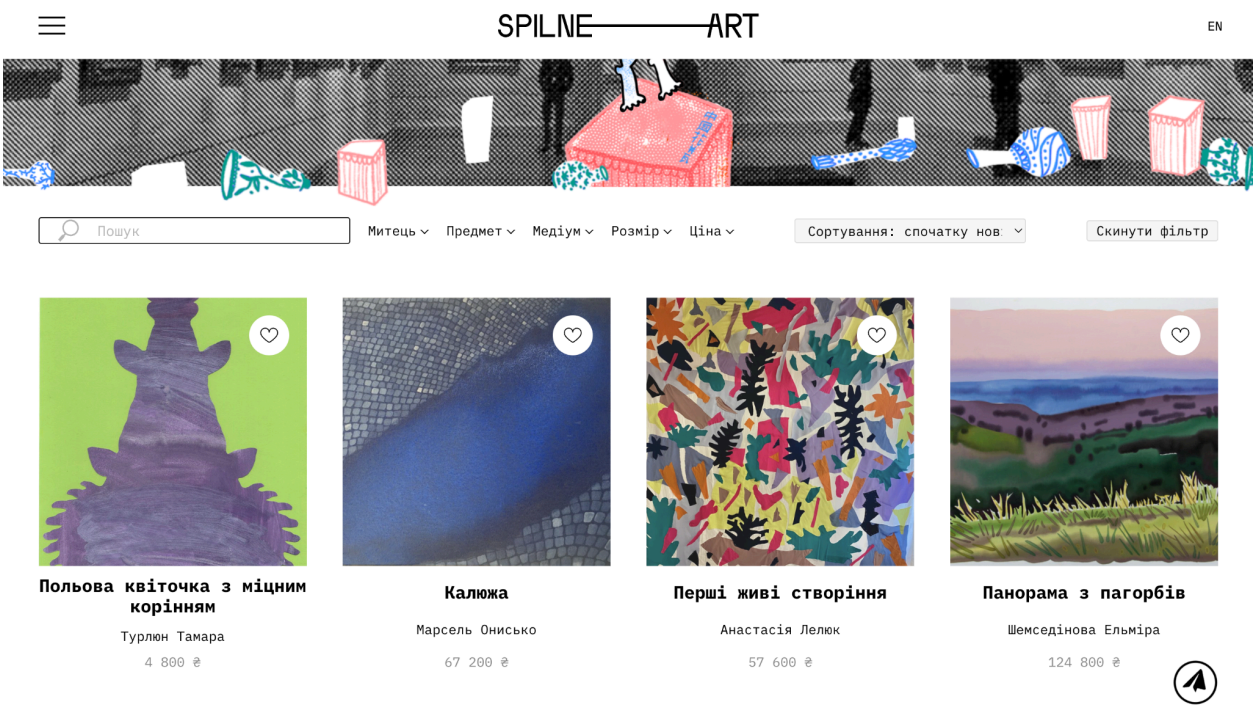

Рисунок 1.2 – Головна сторінка вебплатформи «Spline Art»

Сильні сторони Spline Art:

1. Багатомовна підтримка UA/EN з переходом між мовами «на льоту» – миттєво, без затримок та перезавантаження;
2. Пошук є деталізованим, розширеним динамічним автозаповненням та фільтром із відсіюванням навіть дрібних відмінностей між результатами (Митець/Предмет/Медіум/Розмір/Ціна);
3. Ієрархічна деталізація метаданих артворку (Митець/Медіум/Предмет/Розмір/Основа/Параметри);
4. Є можливість збільшити/наблизити кожен артворк;
5. Наявна рекомендація схожих робіт;
6. Використання scroll-driven анімацій або історій, де елементи вебсайту – лого та ін. – мають візуальний ефект трансформації, залежної від глибини скролу;
7. При наведенні курсору, зображення демонструють zoom-ефект;
8. Деякі елементи реагують на курсор, коли той заходить у їх зону і/або опиняється поруч з ними;
9. Респонсивний дизайн.

Слабкі сторони Spline Art:

1. Валюта є прив’язаною до мовного перемикача UA/EN: якщо активний UA, то всі ціни конвертуються у гривні, якщо EN – у євро. Іншими словами, немає елементів UI для негайної зміни валюти незалежно від будь-якого мовного контексту;
2. Зображення артворку статично кадрується під квадрат; деталі знаходяться поза кадром до відкриття zoom;
3. Відсутність рейтингу митців та відгуків про розпродані артворки;
4. Spline Art «запам’ятовує» користувачів лише тимчасово – через cookies. Якщо cookies будуть видалені, вішліст анулюється.

Behance [3] – Adobe-орієнтована платформа, створена для демонстрації портфоліо, пошуку й налагодження професійних контактів, рекрутингу, а також монетизації креативних послуг (Рисунок 1.3).

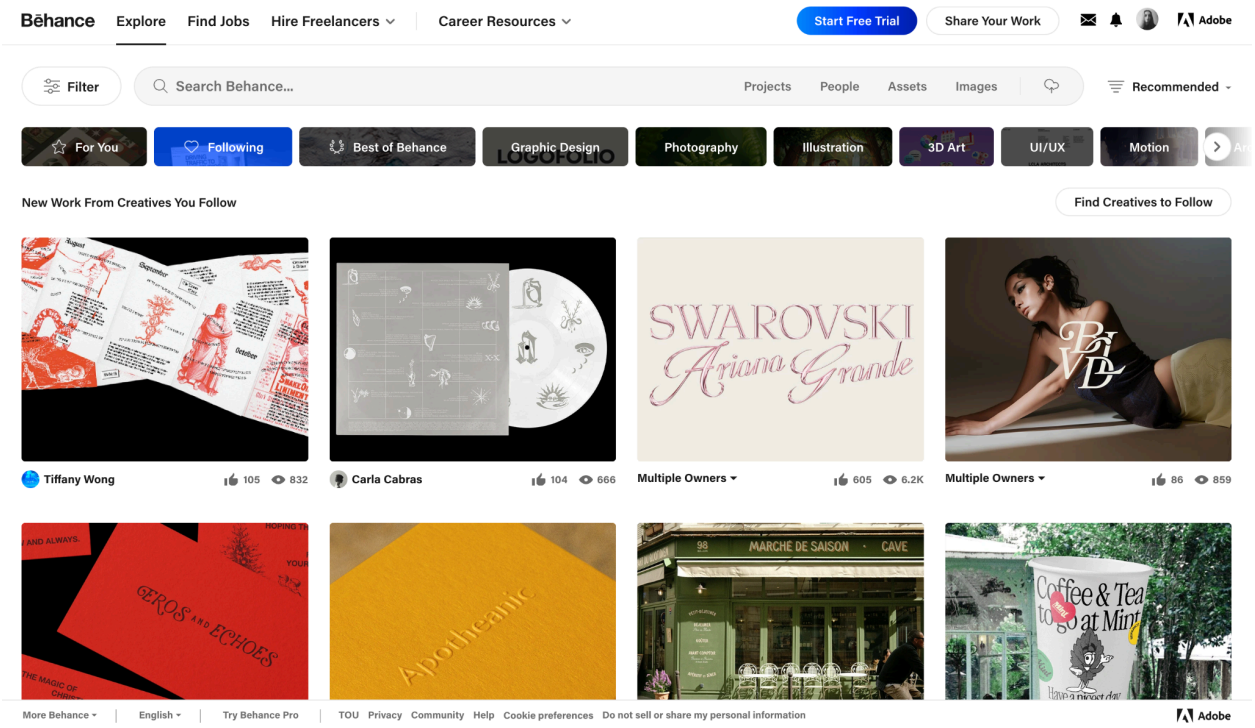

Рисунок 1.3 – Сторінка «Explore» вебплатформи «Behance»

Сильні сторони Behance:

1. Availability автоматично відображає банер із «Available for Hire», який інформує потенційних клієнтів про можливість співпраці;
2. Метрики профілю розмежовані на Project Views та Appreciations (обидві вираховуються сумарно за всі проєкти)/Followers/Following;
3. Інформація профілю надає контроль над базовими секціями, такими як Basic Information/Teams (спільноти Behance, які можна створити і/або до яких можна приєднатися після розгляду заявки)/On the Web (посилання на соцмережі)/та ін., а також кастомними, створеними самим користувачем;

4. Проєкти можна створювати, публікувати, оновлювати без втрати URL-адреси;
5. Behance надає користувачеві виняткову свободу при створенні проєкту – від вибору форматів контенту до побудови логіки його подачі;
6. Взаємодія з проєктом з боку іншого користувача відбувається через Views/Appreciations/коментарі/Moodboards, з боку Behance – тематичні стрічки (відповідно до категорії розміщення самого проєкту), а також згадування у Best of Behance;
7. Персоналізовані рекомендації;
8. Проєкти представлені у вигляді чіткої карткової ґрід-системи;
9. Тонкі градієнтні ефекти та мікровзаємодія в сукупності своїй створюють відчуття глибини;
10. Респонсивний дизайн.

Слабкі сторони Behance:

1. При завантаженні, наприклад, відеоматеріалів платформа застосовує алгоритми стиснення, знижуючи вихідну якість;
2. Перенасиченість платформи призводить до коливань в якості представлених проєктів, що, в свою чергу, знижує шанси окремих проєктів отримати необхідну видимість.

Проаналізувавши Artsy, Spline Art та Behance, можна дійти висновку, що функціональні та інтерфейсні патерни платформ збігаються: кожна з них пропонує механіку детального, багатогранного пошуку; всі вони використовують персоналізовані рекомендації на основі користувацьких поведінкових даних; їхні інтерфейси є або багатомовними, або локально адаптованими; їхні візуальні системи насамперед спираються на збалансований негативний простір із широкомасштабними медіаелементами й продуманими мікровзаємодіями; всі вони забезпечують повністю респонсивний досвід.

2 КОНЦЕПТУАЛЬНИЙ ОГЛЯД

2.1 Принципи візуальної складової

Стратегічну вагу при створенні будь-якого бренду має айдентика. Саме вона повідомляє світові про те, ким є бренд – миттєво і неодноразово.

$$\text{Повторення} \rightarrow \text{згадування} \rightarrow \text{конверсія}, \tag{2.1}$$

де чим *частіше* з'являтимуться складові айдентики, тим *швидше* авдиторія створюватиме асоціації з брендом.

Так, «великою трійцею» айдентики вважаються назва бренду, логотип та кольори.

Як мистецтво, так і сам процес його кураторства – це складна, багатошарова конструкція. Натхненний вірою в те, що сенс кожного артворку видніється поступово, LYR – похідна від «LaYeR» – підкреслює важливість розкриття глибших інтерпретацій, шар за шаром, за межами перших вражень.

При створенні символу головною метою було втілити три ідеї – абстракцію літери «L» як ініціалу LYR, «скульптурну» присутність через монолітну, але водночас різьблену форму, та більш глибокі концептуальні нашарування.

Як результат, конструкція символу LYR, побудована на основі ґріду 7×7, кругів та прямих, досягає гарного балансу між суворою геометрією та динамічною формою (Рисунок 2.1).

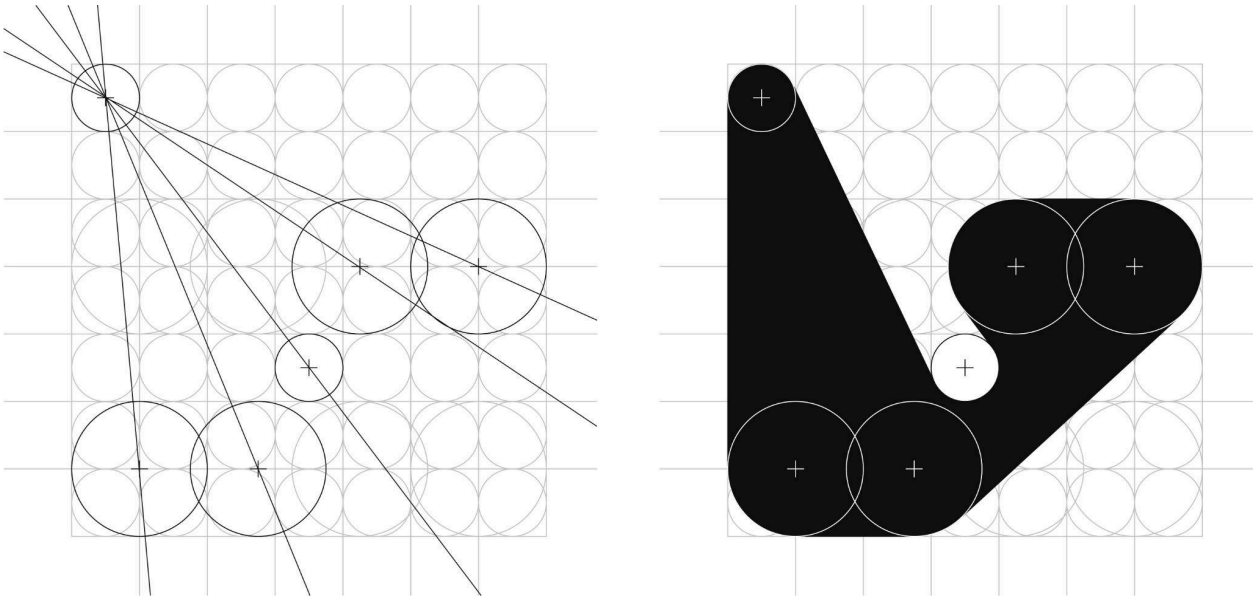

Рисунок 2.1 – Конструкція символу LYR

Основним шрифтом LYR є Helvetica, геометричний sans-serif, який був оптично вдосконалений за допомогою модифікацій міжрядкової висоти та міжлітерного інтервалу (Рисунок 2.2).

Рисунок 2.2 – Типографіка LYR

Повний логотип LYR поєднує в собі символ і назву. Форми літер використовують Helvetica Bold. Однак для того, аби забезпечити візуальну рівновагу в комунікаціях, слід дотримуватися правил мінімального вільного простору навколо логотипу. Для LYR таким мінімальним вільним простором є проміжок між символом і назвою (Рисунок 2.3).

Рисунок 2.3 – Конструкція повного логотипу LYR, визначення мінімального вільного простору

Наостанок фавайкон [4], крихітна емблема, відображається браузерами скрізь, де з'являтиметься назва LYR. Створивши різнорозмірні, але водночас pixel-perfect різноформатні файли, такі як ICO, PNG та SVG, LYR узгоджуватиме єдину, «офіційну» версію брендованого логотипу на всіх можливих цифрових майданчиках. Фавайконом LYR є його символ з м'якими, трохи розсіяними тінями (Рисунок 2.4).

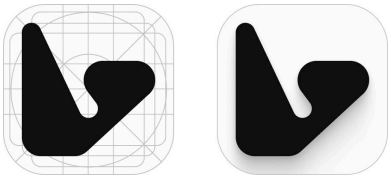

Рисунок 2.4 – Конструкція фавайкону LYR

Маючи такі основи, як назва та логотип, можна підбирати палітру кольорів, аби досягти максимального впливу у всіх точках контакту LYR із зовнішнім світом.

Основу палітри кольорів LYR складають білі, сірі та чорні відтінки (Рисунок 2.5).

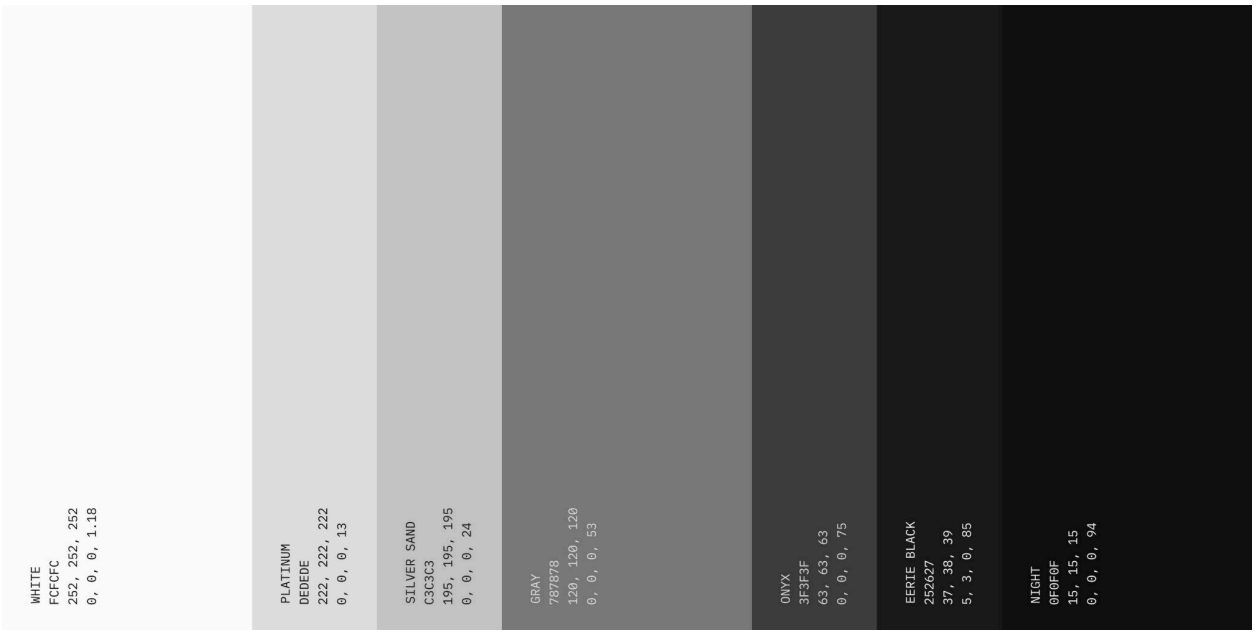

Рисунок 2.5 – Палітра кольорів LYR

Підсумовуючи вищенаведене, синергія між назвою, логотипом та палітрою кольорів не тільки посилюватиме сильні сторони кожного з них, але й формуватиме цілісну та, найголовніше, запам'ятовувану ідентичність LYR.

2.2 Діаграма сегментів таргету

Будь-який бренд – це узгоджена система рішень, яка працює тільки тоді, коли ґрунтується на чітко визначеному таргеті – цільовій авдиторії. Таргет сучасного цифрового мистецтва можна умовно поділити на три сегменти (Рисунок 2.6):

1. Незалежні митці/кині – користувачі LYR: ті, хто тільки починає творити, а також досвідчені професіонали. В першу чергу, LYR націлений на підтримку незалежних митців/кинь та надання їм можливості бути почутими та визнаними, впізнаваними. Іншими словами, LYR – спільнота митців/кинь для митців/кинь;
2. Куратори музеїв, артгалерей, виставкових просторів і/або ін. – гості LYR. Завдяки тому, що LYR планується як осередок цифрового мистецтва, попередньо відібраного власними кураторами, зовнішні куратори зможуть:

2.1. Відслідковувати нові тенденції;

2.2. Кожен допис буде належним чином структуровано, а отже, куратори отримають глибше розуміння про творчий процес та мислення митця/кині;

2.3. Як наслідок, знаходитимуть талановитих митців/кинь; через лінки на соцмережі встановлюватимуть з ними коннект для подальшої кооперації;

2.4. Зможуть уявити, як артворк може бути інтегрований в ту чи іншу фізичну локацію.

3. Зацікавлена авдиторія – ще одні гості LYR.

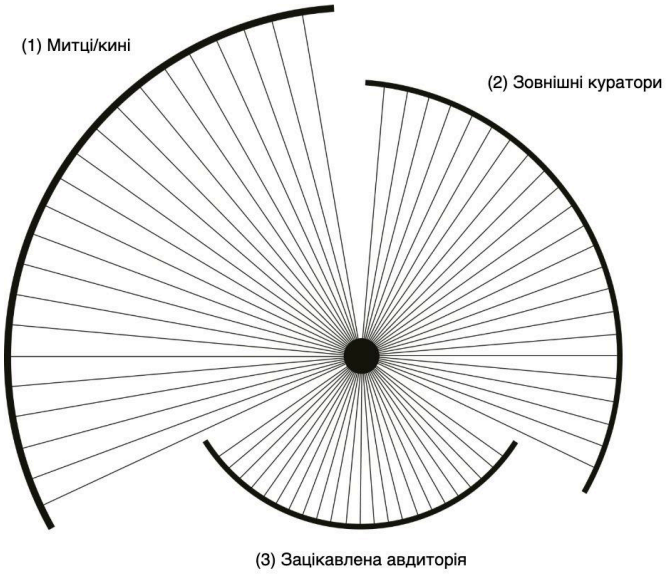

Рисунок 2.6 – Діаграма сегментів таргету LYR

LYR свідомо надає перевагу (1) митцям/киням, (2) зовнішнім кураторам та (3) зацікавленій авдиторії – і тільки в такому порядку. Обґрунтувати таку пріоритезацію можна таким чином:

1. Без митців/кинь LYR як ініціатива не матиме культурного продукту – тож їхній успіх є успіхом LYR;
2. Зовнішні куратори знаходяться між митцями/кинями та інституціями. Коли зовнішні куратори можуть ефективно шукати

таланти, вони створюють тристоронній ефект: митці/кині отримують розголос, зовнішні куратори – нові проєкти, а LYR – довіру;

3. Увага авдиторії має важливе значення для культурного впливу, але вона слідує за пропозицією – артворками – та професійною підтримкою.

2.3 Розробка моделі опенколу

LYR шукав модель, за якої:

1. Кожен поданий артворк проходив би через очі та вдумливий, критичний відбір внутрішніми кураторами платформи – командою, чий спільний досвід формує та спрямовує мистецьку траєкторію LYR;
2. Сам процес відбору залишався би прозорим, доступним та керованим чіткими критеріями.

Єдиною моделлю, яка задовольняє обидві вимоги, є модель опенколу – публічного запрошення для митців/кинь подавати артворки в наступних рамках:

1. Створити артворк; зібрати медіаматеріали;
2. Заповнити опенкол:
    2.1. Вказати тайтл – назву – артворку;
    2.2. Сформувати сніпет (≤ 20 слів), який миттєво «зчитує» суть артворку;
    2.3. Обрати один з двох «шарів» LYR, а саме – креативне кодування або візуальне програмування;
    2.4. Описати артворк у розгорнутому стейтменті (≥ 100 слів);
    2.5. Вирішити, чи дозволити редакторську підтримку від команди LYR;

3. Отримати повідомлення на пошту і/або в межах LYR з результатами відбору (куратори LYR розглядають заявки впродовж 2-5 робочих днів);
4. За умови відхилення, куратор зазначає причину; за умови схвалення, публікація артворку відбувається миттєво;
5. Отримати повідомлення на пошту і/або в межах LYR зі сповіщенням про публікацію артворку з його URL-адресою.

Послідовність дій, показана на рисунку 2.7, надає уявлення про процес, якого має дотримуватися митець/киня, подаючись на опенкол. Редагування раніше опублікованого допису відбувається через той самий опенкол, але з дещо зміненою структурою: форма опенколу підтягує вже існуючий допис, де нижче митець/киня може вказати, що саме його/її не влаштовує. Однак опенкол все одно має пройти стандартний, але тепер вже повторний розгляд правок.

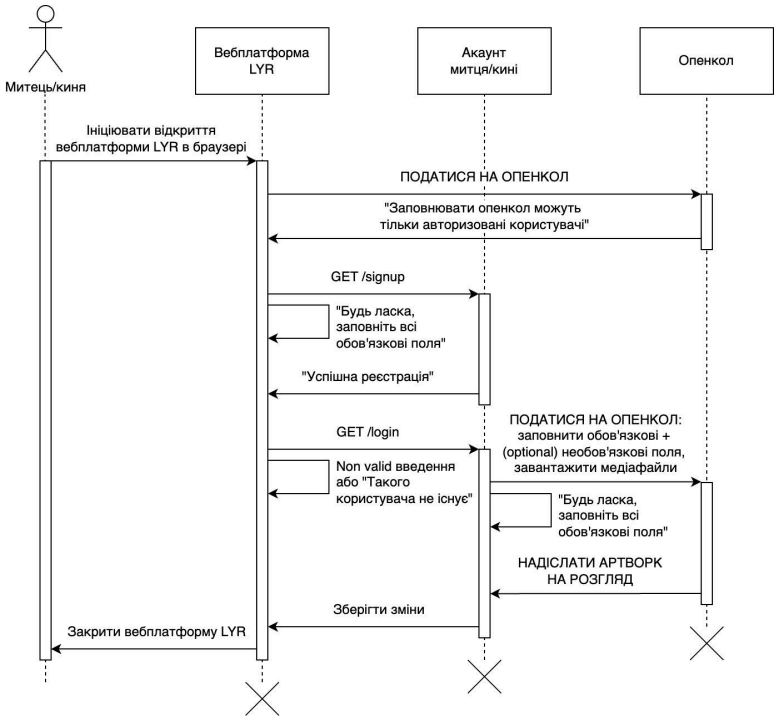

Рисунок 2.7 – Діаграма послідовності (sequence diagram) подання артворку на опенкол для митця/кині

Натомість робочий процес куратора LYR окреслено на рисунку 2.8.

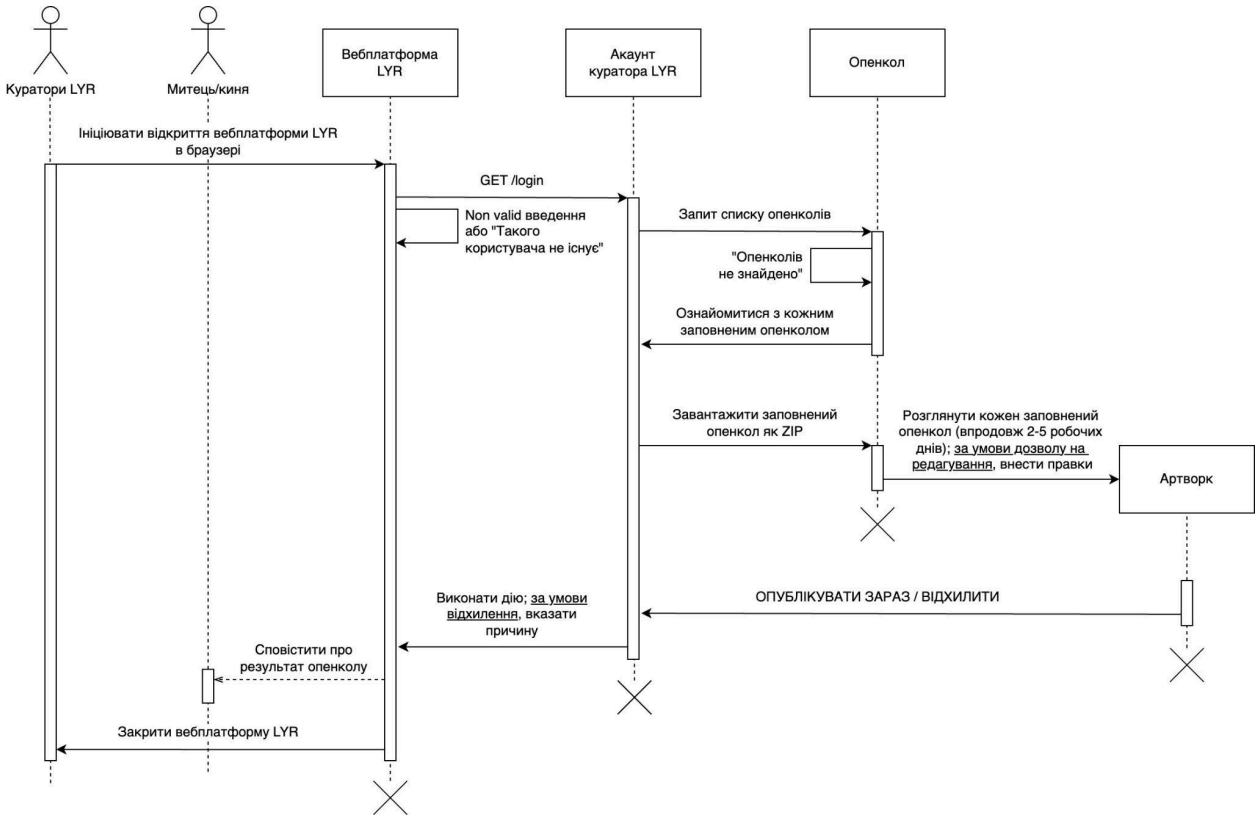

Рисунок 2.8 – Діаграма послідовності (sequence diagram) схвалення/відхилення опенколів для кураторів LYR

У підсумку, модель опенколу гарантуватиме, що кожен артворк буде належним чином контекстуалізований та повністю вписаний в бачення LYR.

2.4 Композиція ґрід-системи та базові UI-макети

Ґріди є як інструментом, так і мовою дизайну. Вони вносять порядок та ритм, підтримують адаптивність макетів, а також їх передбачуваність. Основними компонентами ґрідів вважаються margins, які можна назвати зовнішніми «безпечними» межами, завдяки яким контент не торкається країв сторінки, та gutters – інтервали між колонками, а іноді й між рядками.

LYR дотримується наступної практики (Рисунок 2.9):

1. Десктопна ґрід-система:

   Grid – 12 Columns

Type – Stretch

Margin – 20 px

Gutter – 20 px

2. Мобільна ґрід-система:

Grid – 4 Columns

Type – Stretch

Margin – 15 px

Gutter – 20 px

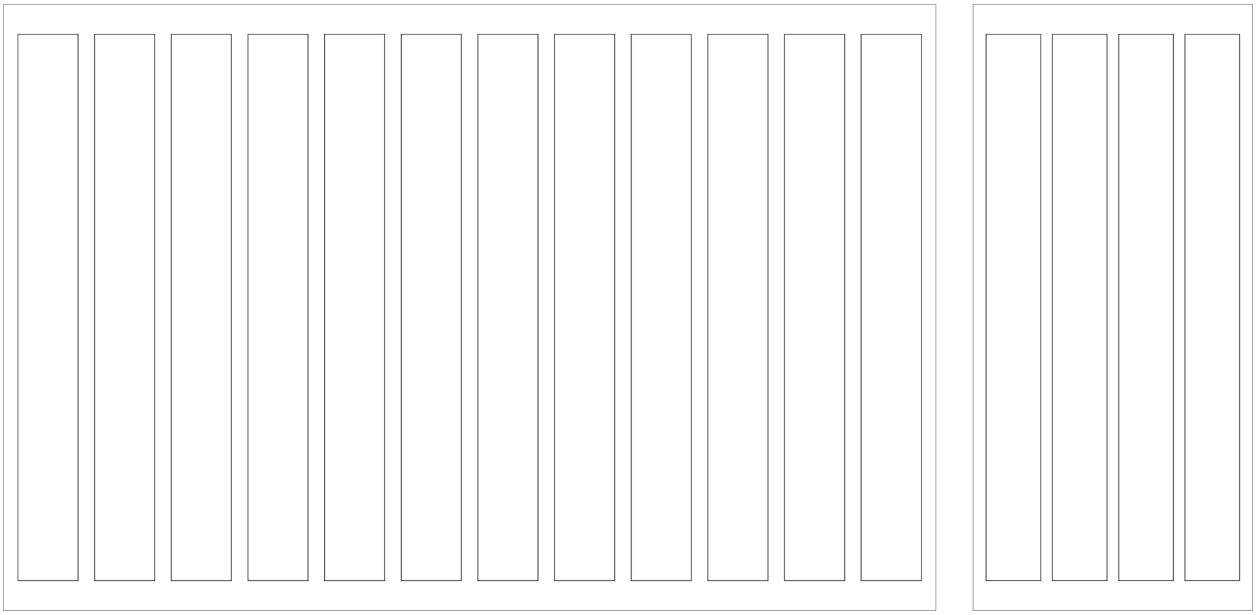

Рис. 2.9 – Ґрід-система LYR

Орієнтуючись на чіткі контури ґрід-системи, стає можливим визначити просторову логіку та створити фундамент для побудови базових UI-макетів вебплатформи.

Макет авторизації (Рисунок 2.10) служить початковою точкою взаємодії з LYR, зосереджуючись на автентифікації, ідентифікації користувача та наданні доступу до акаунту. Зазвичай, така форма використовує базові елементи, такі як 1 – поля введення імейлу користувача та пароля; 2 – тригер процесу авторизації, а також можливості відновлення пароля та реєстрації.

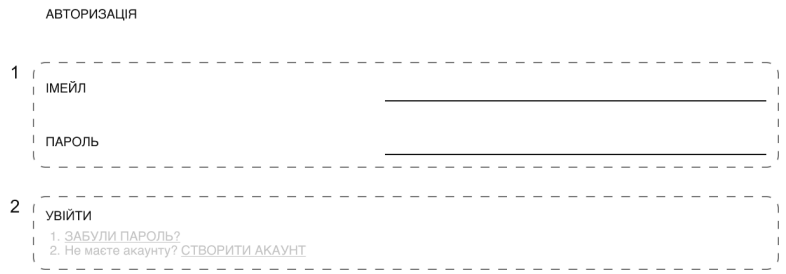

Рисунок 2.10 – Макет авторизації LYR

На відміну від авторизації, макет реєстрації (Рисунок 2.11) наслідує функціонал першої, але також і вводить деякі додаткові елементи, такі як 1 – поля введення персональних даних (імені та прізвища митця/кині), поля введення імейлу користувача та пароля з вимогами до його створення; 2 – тригер процесу реєстрації, можливість входу в акаунт, якщо він вже існує на момент реєстрації, а також попереднє ознайомлення з Privacy Policy та Terms and Conditions платформи LYR.

Рисунок 2.11 – Макет реєстрації LYR

На рисунку 2.12 зображено базовий макет вебінтерфейсу LYR, де 1 – навбар з тригерами на інші елементи; 2 – блок негативного простору; 3 – контейнер із динамічним контентом, зокрема футером.

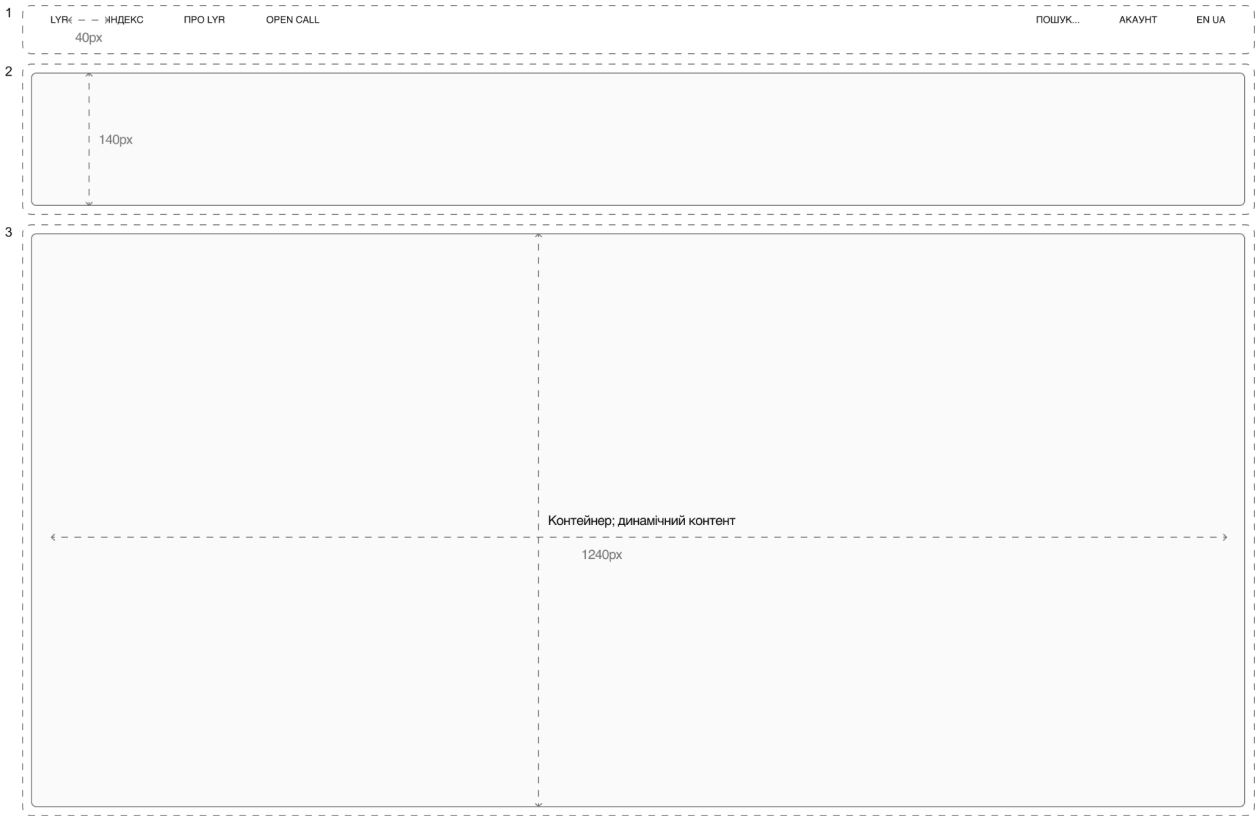

Рисунок 2.12 – Базовий макет вебінтерфейсу LYR

Варто також зазначити, що у контексті моделювання вебінтерфейсів первинні макети являють собою початкові концепції дизайну, які можуть зазнавати змін або певної адаптації під час процесу розробки.

3 РЕАЛІЗАЦІЯ LYR

3.1 Технологічний стек, інструменти

Фронтенд-стек LYR спирається на мейнстрімні мови програмування з наступних причин:

1. HTML [5] задаватиме «змістовний» сенс про вміст сторінок за допомогою таких тегів, як <header>, <nav>, <main>, <footer> та ін.;
2. CSS [6] надасть змогу розбити стилі на кілька файлів за логікою HTML-структури та за допомогою @import зібрати їх в один єдиний style.css. Маючи кольорову палітру LYR, в :root можна буде прописати CSS-змінні для кольорів, зокрема параметрів ґріду, відступів, базових позиціонувань та ін.;
3. Чистий JavaScript (JS) [7] забезпечуватиме легку динаміку мікровзаємодій та використання AJAX.

Натомість для бекенд-стеку LYR обирає:

1. Node.js, бо одна мова – JS – як у фронтенді, так і в бекенді; Express, розділятиме API на окремі модулі в routes/ [8];
2. NPM [9] з його величезною екосистемою пакетів дозволить працювати з хешуванням паролів (bcrypt), файлами (multer) та архівами (archiver).

Довести, наскільки вдалим був вибір нинішнього стеку LYR, можна, подивившись на результати опитування 2024 Stack Overflow Developer Survey [10]: JavaScript та HTML/CSS й досі лишаються найпопулярнішими мовами (~62% та ~53% відповідно), Node.js (~41%) та Express (~18%) займають топ-5 вебтехнологій, тоді як NPM (~50%) на 2 місці серед інших інструментів.

Що стосується бази даних, то вибір припав на PostgreSQL [11] – гарно протестовану separate-server архітектуру з її багатим фреймворком розширень та не менш потужним SQL-арсеналом. Менеджити таку БД можна, скориставшись синергією DBngin [12] та TablePlus [13]: перший інструмент

здатен встановлювати, запускати або зупиняти інстанси, керувати кількома версіями PostgreSQL, тоді як другий – підключатися до будь-яких інстансів, переглядати схеми БД, виконувати запити та редагувати дані через гнучкий GUI. Обидва інструменти розроблені однією командою, існують в одній екосистемі, а отже, зручність є не стільки умовною, скільки практичною – DBngin напряму створює коннект з TablePlus з відповідним хостом, портом, користувачем (postgres) і паролем (за замовчуванням postgres), що забезпечує набагато швидший старт.

Visual Studio Code (VS Code) [14] є кросплатформним IDE, доволі простим та плавним. Завдяки активному маркетплейсу розширень, VS Code поєднує в собі все необхідне для того, аби підлаштувати свій робочий простір майже під будь-яку задачу.

VS Code також має вбудовану підтримку Git [15]. Після створення директорії проєкту, вона ініціалізується як новий Git-репозиторій у секції Source Control. Всі файли додаються до стейджу – проміжного етапу перед фіксацією змін у Git.

Відповідно до того самого опитування 2024 Stack Overflow Developer Survey [10], PostgreSQL (~49%) та VS Code (~74%) утримують за собою топ-1. А отже, підсумувавши, технології та інструменти, обрані LYR, відповідають сучасним стандартам та стабільно зберігають на сьогодні першість у сфері розробки.

3.2 Моделювання бази даних

Оскільки LYR працює з PostgreSQL, в директорії було створено декілька файлів – .env, db.js та schema.sql:

1. .env зберігає конфіденційні налаштування та параметри у форматі KEY=VALUE. Вноситься до .gitignore, аби в подальшому файл не комітився до репозиторію та випадково не розкривав чутливі дані;
2. db.js налаштовує та ініціалізує пул з lyr_db, використовуючи вміст .env;

3. schema.sql описує структуру БД – таблиці, індекси, обмеження та ін. – щоб і надалі існувала можливість відтворити її при деплої або міграціях БД.

Моделювання lyr_db відбувалось безпосередньо в самому TablePlus, після чого всі таблиці були експортовані як один єдиний SQL-запит до schema.sql. Так, в lyr_db є такі ключові таблиці, як users, artworks та artwork_likes, а також прописана для останньої PL/pgSQL-функція like_artwork.

Таблиця users зберігає інформацію про зареєстрованих митців/кинь (Рисунок 3.1).

```
CREATE TABLE "public"."users" (
    "id" int4 NOT NULL DEFAULT nextval('users_id_seq'::regclass),
    "first_name" text NOT NULL,
    "last_name" text NOT NULL,
    "email" text NOT NULL,
    "password_hash" text NOT NULL,
    "is_admin" int4 NOT NULL DEFAULT 0,
    "bio" text,
    "contacts" text,
    "created_at" timestamptz NOT NULL DEFAULT now(),
    PRIMARY KEY ("id")
);
```

Рисунок 3.1 – SQL-запит на створення таблиці users, schema.sql

Таблиця artworks містить дані про кожен з артворків, що був знайдений в межах LYR – тільки-но поданий на опенкол, відхилений і/або той, що вже опублікований (Рисунок 3.2).

```
CREATE TABLE "public"."artworks" (
    "id" int4 NOT NULL DEFAULT nextval('artworks_id_seq'::regclass),
    "user_id" int4,
    "title" text NOT NULL,
    "artist_name" text NOT NULL,
    "phone" text,
    "socials" text,
    "layer" text NOT NULL,
    "snippet" text,
    "media_paths" _text NOT NULL DEFAULT '{}'::text[],
    "statement" text,
    "feedback_allowed" bool NOT NULL DEFAULT true,
    "editing_allowed" bool NOT NULL DEFAULT false,
    "published_at" date,
    "likes_count" int4 NOT NULL DEFAULT 0,
    "created_at" timestamptz NOT NULL DEFAULT now(),
    "reject_note" text,
    "status" text NOT NULL DEFAULT 'РОЗГЛЯДАЄТЬСЯ'::text,
    PRIMARY KEY ("id")
);
```

Рисунок 3.2 – SQL-запит на створення таблиці artworks, schema.sql

Таблиця artwork_likes призначена для фіксування метрики «подобається» для кожного з артворків з боку користувачів (Рисунок 3.3).

```
CREATE TABLE "public"."artwork_likes" (
    "artwork_id" int4 NOT NULL,
    "ip_address" inet NOT NULL,
    "liked_at" timestamptz NOT NULL DEFAULT now(),
    "user_id" int4 NOT NULL,
    PRIMARY KEY ("artwork_id","user_id")
);
```

Рисунок 3.3 – SQL-запит на створення таблиці artwork_likes, schema.sql

PL/pgSQL-функція like_artwork керує каунтером likes_count в таблиці artworks (Рисунок 3.4).

```
CREATE
OR REPLACE FUNCTION public.like_artwork (p_artwork integer, p_ip inet, p_user integer) RETURNS
void LANGUAGE plpgsql AS $function$
BEGIN
    INSERT INTO artwork_likes (artwork_id, ip_address, user_id)
      VALUES (p_artwork, p_ip, p_user);
    UPDATE artworks
       SET likes_count = likes_count + 1
     WHERE id = p_artwork;
EXCEPTION
    WHEN unique_violation THEN
        NULL;
END;
$function$
```

Рисунок 3.4 – SQL-запит на створення PL/pgSQL-функції like_artwork, schema.sql

Візуально уявити структуру lyr_db також можна, представивши її схематично за допомогою діаграми (Рисунок 3.5). Ретельно окреслюючи атрибути, їхні відповідні типи даних та зв’язки між таблицями в цілому, схема задасть вектор, який обходитиме ризики виникнення аномалій при роботі з даними.

Рисунок 3.5 – Діаграма схематичного представлення БД lyr_db

Моделювання бази даних – це про ретельно продуманий процес, що закладає фундамент всього проєкту. Завчасно інвестувати час в цей процес означає уникнути дорогих – і часто руйнівних – наслідків поганої структури, які доведеться виправляти «на ходу» вже в продакшені.

3.3 Реєстрація

Спочатку форма реєстрації перевіряє всю необхідну інформацію – ім’я та прізвище, імейл, пароль – на наявність введених даних. Поля, які є обов’язковими для заповнення, в лейблі позначені *, тоді як в інпуті – атрибутом required (Рисунок 3.6).

```
<div class="form-rows">
  <div>
    <label class="form-label" for="signup-password">ПАРОЛЬ *</label>
    <p class="form-note" style="margin-top: 5px;">
      1. Підтримуються усі друковані ASCII + пробіл.<br>
      2. Автоматично відхилятимуться шаблонні паролі, такі як «123456», «password» тощо.<br>
      3. Ми радимо створювати довгі пасфрази (≥ 4 слів).
    </p>
  </div>
  <input class="form-control" type="password" id="signup-password" name="password"
minlength="6"
    pattern="[ -~]+" autocomplete="new-password" style="align-self: start;" required>
</div>
```

Рисунок 3.6 – Обов’язкове для заповнення поле ПАРОЛЬ

Для пароля було побудовано декілька ліній оборони, перша з яких – на стороні клієнта, що відповідає за UI, початкові валідації, пакування даних в JSON та їхню відправку через fetch.

Так, form-note – підказка, зображена на рисунку 3.6 – інформує користувача та слугує детальним описом ряду перевірок, які виконуються на стороні клієнта перед відправкою форми (Рисунок 3.7).

```
const pass = passInput.value;
if (!/^[\x20-\x7E]+$/.test(pass)) {
  ...
}
const PASS_MIN_LENGTH = 6;
if (pass.length < PASS_MIN_LENGTH) {
  ...
}
if (blackList.includes(pass.toLowerCase())) {
  ...
}
const PASS_MIN_WORDS = 4;
const wordCount = pass.trim().split(/\s+/).filter(Boolean).length;
if (wordCount < PASS_MIN_WORDS) {
  ...
}
```

Рисунок 3.7 – Ряд перевірок для пароля на стороні клієнта

Друга лінія оборони – на стороні сервера, що відповідає здебільшого за фінальну валідацію, безпечне хешування, з’єднання з БД, а також формування відповіді з коректним статусом і/або повідомленням.

Найбільш безпечний спосіб зберігання пароля – перетворити його на дані, які неможливо інтерпретувати назад у вихідний пароль. Досягти цього можна за допомогою хешування пароля алгоритмом bcrypt з оптимальними 12 раундами: зі збільшенням кількості раундів зростає стійкість до брутфорсу

– перебору всіх можливих варіантів, доки не буде знайдено потрібний – проте й збільшується час хешування. Важливо також зазначити, що в БД зберігається лише версія password_hash, ніколи – сам пароль (Рисунок 3.8).

```
const HASH_ROUNDS = 12;
...
const hash = await bcrypt.hash(password, HASH_ROUNDS);
```

Рисунок 3.8 – Хешування пароля алгоритмом bcrypt

Відтак у наступному фрагменті, зображеному на рисунку 3.9, pool.query намагається виконати запит на додавання нового користувача до БД. Сам запит INSERT в поєднанні з унікальним індексом на стовпець email таблиці users діють як єдине ціле та гарантують унікальність імейлу для кожного нового користувача. Всі нюанси перехоплюються й виводяться в console.error, інакше – успішне додавання, відповідь 201 з JSON виду {id, created_at} та переадресація на авторизацію.

```
router.post('/register', async (req, res) => {
  const { first_name, last_name, email, password } = req.body;
  ...

  try {
    ...
    const { rows } = await pool.query(
      `INSERT INTO users (first_name, last_name, email, password_hash)
      VALUES ($1,$2,$3,$4)
      RETURNING id, created_at`,
      [first_name.trim(), last_name.trim(), email.toLowerCase(), hash]
    );
    res.status(201).json({ id: rows[0].id, created_at: rows[0].created_at });
  } catch (err) {
    ...
  }
});
```

Рисунок 3.9 – pool.query на INSERT користувача

Зрештою процес реєстрації захищений від найпоширеніших ризиків – простий пароль, SQLi, злам хешів та ін. – залишаючи водночас і простір для подальших вдосконалень.

3.4 Авторизація

Форма авторизації обмежується обов’язковими для заповнення полями ІМЕЙЛ та ПАРОЛЬ: спершу шукаємо імейл серед користувачів в БД, потім порівнюємо bcrypt-хеші та повертаємо {id, is_admin} (Рисунок 3.10).

```
router.post('/login', async (req, res) => {
  const { email, password } = req.body;
  ...

  try {
    const { rows } = await pool.query(
      'SELECT id, password_hash, is_admin FROM users WHERE email = $1',
      [email.toLowerCase()]
    );
    ...

    const ok = await bcrypt.compare(password, rows[0].password_hash);
    ...
    res.json({ id: rows[0].id, is_admin: rows[0].is_admin });
  } catch (err) {
    ...
  }
});
```

Рисунок 3.10 – pool.query на авторизацію користувача

Якщо ж було спіймано негативну відповідь, виведеться «розмита» причина, без деталізації, що саме виявилось неправильним (Рисунок 3.11).

```
else {
  const { message } = await r.json().catch(() => ({}));
  alert(message || 'Невірні облікові дані.');
}
```

Рисунок 3.11 – alert message за умови негативної відповіді

Після авторизації залогіненого користувача буде переадресовано на сторінку ІНДЕКС.

3.5 Акаунт

Враховуючи, що основною метою акаунту є миттєво відображати динаміку в даних митця/кині, вся функціональність була об’єднана в єдиному компоненті – модальному вікні, що існує в трьох станах та, залежно від ролі користувача, показує/приховує ту чи іншу секцію:

1. Митець/киня бачить readonly-стан АКАУНТ та edit-стан НАЛАШТУВАННЯ;
2. Куратор LYR додатково отримує admin-стан ОПЕНКОЛИ.

Видимість кожної з секцій контролюється тригерами, розташованими в хедері, через три JS-функції – showAccount(), showSettings(), showPending(), які показують/приховують потрібну <section> (Рисунок 3.12).

```
function showAccount() {
  viewAccount.classList.remove('hidden');
  viewSettings.classList.add('hidden');
  viewPending.classList.add('hidden');
}
function showSettings() {
  viewAccount.classList.add('hidden');
  viewSettings.classList.remove('hidden');
  viewPending.classList.add('hidden');
}
function showPending() {
  viewAccount.classList.add('hidden');
  viewSettings.classList.add('hidden');
  viewPending.classList.remove('hidden');
  loadPending();
}
```

Рисунок 3.12 – Функції перемикання видимості для секцій view-account, view-settings, view-pending

Найперше, що витягується – користувач (Рисунок 3.13).

```
...
const uid = +req.params.id;
const { rows: u } = await pool.query(
  `SELECT first_name, last_name,
  email, bio, contacts
  FROM users WHERE id = $1`,
  [uid]
);
...
```

Рисунок 3.13 – pool.query на SELECT користувача

За користувачем витягуються всі артворки, нові – першими. /:id/dashboard є ендпоінтом, що формує та повертає один зведений JSON (Рисунок 3.14).

```
...
const { rows: a } = await pool.query(
  `SELECT id, title, media_paths, snippet,
  published_at, reject_note, status, likes_count
  FROM artworks
  WHERE user_id = $1
  ORDER BY created_at DESC`,
  [uid]
);
res.json({
  first_name: u[0].first_name,
  last_name: u[0].last_name,
  email: u[0].email,
  bio: u[0].bio,
  contacts: u[0].contacts,
  artworks: a
});
```

Рисунок 3.14 – pool.query на SELECT всіх артворків користувача та формування зведеного JSON

Єдиною точкою, яка синхронізує всі секції модального вікна, є loadDashboard(). Шаблонна функція addGridRow() рендерить ґріди для списку опублікованих артворків в АКАУНТ, повного списку артворків в НАЛАШТУВАННЯ, а також для loadPending() в ОПЕНКОЛИ для куратора LYR (Рисунок 3.15).

```
function addGridRow(container, { idx, title, snippet, statusText, id }) {
  const row = document.createElement('div');
  row.className = 'grid';
  row.innerHTML = `
    <div class="col col1"><div class="line"></div><div class="grid-num">${idx + 1}</div></div>
    <div class="col col2"><div class="line"></div><div class="grid-title">${title}</div></div>
    <div class="col col3"><div class="line"></div><div class="grid-text">${snippet || '—'}</
div></div>
    <div class="col col4"><div class="line"></div><div class="grid-text">${statusText}</div></
div>`;
  ...
  container.appendChild(row);
}
```

Рисунок 3.15 – Шаблонна функція addGridRow()

З налаштувань митець/киня має можливість відредагувати такі поля, як ПОВНЕ ІМ’Я – об’єднання полів ІМ’Я та ПРІЗВИЩЕ – ПАРОЛЬ, ПРО МИТЦЯ/КИНЮ, контактні ЛІНКИ. Всі поля, окрім зміни пароля, використовують автозаповнення, але, порівняно з формою опенколу, не мають атрибута readonly. Форма підтримує часткове редагування: у запиті

передаються виключно ті поля, які користувач фактично змінив, без примусового визначення інших полів як обов’язкових.

Якщо вводиться старий + новий пароль, вони порівнюються алгоритмом bcrypt.compare(), після чого новий пароль хешується bcrypt.hash() з тим самим HASH_ROUNDS – захист нового пароля повинен нічим не поступатися захисту старого (Рисунок 3.16).

```
const match = await bcrypt.compare(old_password, rows[0].password_hash);
...
const newHash = await bcrypt.hash(new_password, HASH_ROUNDS);
```

Рисунок 3.16 – Порівняння старого і нового паролів алгоритмом bcrypt.compare, хешування нового пароля bcrypt.hash

Нижче митець/киня бачить ґрід-список з усіма своїми артворками – опублікованими, тими, що перебувають на розгляді, та відхиленими – з поточним статусом:

1. РОЗГЛЯДАЄТЬСЯ – дефолтний статус;
2. ОПУБЛІКОВАНО із датою публікації published_at у форматі DD.MM.YYYY;
3. ВІДХИЛЕНО із приміткою reject_note, де вказано причину відмови.

Додатково, якщо артворк схвалено, поруч з датою публікації фіксується кількість вподобань (Рисунок 3.17).

```
if (a.status === 'ОПУБЛІКОВАНО') {
  statusText = `ОПУБЛІКОВАНО ${formatDate(a.published_at)}, ПОДОБАЄТЬСЯ: ${a.likes_count}`;
}
```

Рисунок 3.17 – Формат статусу СХВАЛЕНО з каунтером ПОДОБАЄТЬСЯ

Для адміністратора – куратора LYR – хедер модального вікна розширюватиметься тригером ОПЕНКОЛИ. Для цього перевіряється поточне значення isAdmin, передане разом з авторизацією користувача (Рисунок 3.18).

```
const isAdmin = localStorage.getItem('isAdmin') === '1';
if (isAdmin) {
  pendingBtn = document.createElement('button');
  pendingBtn.className = 'pending-btn';
  pendingBtn.textContent = 'ОПЕНКОЛИ';
  rightControls.prepend(pendingBtn);
  pendingBtn.addEventListener('click', showPending);
}
```

Рисунок 3.18 – Тригер ОПЕНКОЛИ

Використовуючи ту саму шаблонну функцію addGridRow(), куратор бачить ґрід-список опенколів, сформований SQL-умовою published_at IS NULL AND reject_note IS NULL (Рисунок 3.19).

```
router.get('/pending', async (_, res) => {
  try {
    const { rows } = await pool.query(
      `SELECT id, title, snippet,
      reject_note, created_at
      FROM artworks
      WHERE published_at IS NULL
      AND reject_note IS NULL
      ORDER BY created_at DESC`
    );
    res.json(rows);
  } catch (err) {
    ...
  }
});
```

Рисунок 3.19 – pool.query на SELECT опенколів на розгляді

Кожен опенкол в ґрід-списку стає selectable, завдяки чому куратор обирає, що саме робити з опенколом – ЗАВАНТАЖИТИ, ОПУБЛІКУВАТИ ЗАРАЗ і/або ВІДХИЛИТИ.

Обравши ЗАВАНТАЖИТИ, формується ZIP, куди пакується текстова версія заповненого опенколу opencall.txt, а також всі прикріплені медіафайли (Рисунок 3.20).

```
router.get('/:id/archive', async (req, res) => {
  const artId = +req.params.id;
  const { rows } = await pool.query(
    `SELECT a.*, u.first_name, u.last_name, u.email
     FROM artworks a
     JOIN users u ON a.user_id = u.id
     WHERE a.id = $1`, [artId]
  );
  ...
  const zip = archiver('zip');
  ...
  zip.append(buildTxt(art, user), { name: 'opencall.txt' });
  art.media_paths.forEach(p => {
    const abs = path.resolve('public', p);
    if (fs.existsSync(abs)) zip.file(abs, { name: path.basename(abs) });
  });
  zip.finalize();
});
```

Рисунок 3.20 – Формування та завантаження опенколу як ZIP

Обравши ОПУБЛІКУВАТИ ЗАРАЗ, статус артворку з РОЗГЛЯДАЄТЬСЯ змінюється на ОПУБЛІКОВАНО + записується дата публікації артворку published_at (Рисунок 3.21).

```
router.patch('/:id/publish', async (req, res) => {
  const artId = +req.params.id;
  const { published_at } = req.body;
  ...
  try {
    await pool.query(
      `UPDATE artworks
      SET published_at = $1,
      status = 'ОПУБЛІКОВАНО'
      WHERE id = $2`,
      [published_at, artId]
    );
    res.sendStatus(204);
  } catch (err) {
    ...
  }
});
```

Рисунок 3.21 – pool.query на UPDATE опублікованого артворку

Обравши ВІДХИЛИТИ, статус артворку записується до reject_note, status змінюється на ВІДХИЛЕНО (Рисунок 3.22)

```
router.patch('/:id/reject', async (req, res) => {
  const artId = +req.params.id;
  const { reject_note } = req.body;
  ...
  try {
    await pool.query(
      `UPDATE artworks
      SET reject_note = $1,
      status = 'ВІДХИЛЕНО'
      WHERE id = $2`,
      [reject_note, artId]
    );
    res.sendStatus(204);
  } catch (err) {
    ...
  }
});
```

Рисунок 3.22 – pool.query на UPDATE відхиленого артворку

Таким чином, модалка для акаунту залишається однією з найзручніших практик через свою гнучкість та швидкість.

3.6 Опенкол

Подаватися на опенкол можуть лише залогінені митці/кині. Щойно відбувається фокус на будь-якому полі, перевіряється наявність userId; якщо немає – переадресація на авторизацію (Рисунок 3.23).

```
function checkLogin(e) {
  if (!uid) {
    e && e.preventDefault();
    location.href = '../login.html';
    return true;
  }
  return false;
}
form.addEventListener('focusin', checkLogin, { once: true });
```

Рисунок 3.23 – Функція перевірки userId

Форму опенколу можна умовно поділити на три секції:

1. Секція даних митця/кині – ім’я та прізвище, імейл, контактний номер телефону, лінки на соцмережі/портфоліо – ідентифікує аплікаанта/ку та надає кураторам канали зв’язку;
2. Секція даних артворку – тайтл, «шар» LYR, сніпет, медіа, розгорнутий стейтмент – максимально деталізує роботу;
3. Дозвіл на професійне редакційне втручання.

Щоб пришвидшити повторне подання, поля ІМ’Я, ПРІЗВИЩЕ та ІМЕЙЛ мають автозаповнення з БД й позначені атрибутом readonly; водночас поле з лінками все ще залишається редагованим, адже посилання можуть постійно змінюватися.

Деякі поля, такі як СНІПЕТ та РОЗГОРНУТИЙ СТЕЙТМЕНТ, доповнені додатковою валідацією введення: каунтер слів підказує користувачу, чи відповідає введений текст встановленим лімітам. Також поле МЕДІА з drag-and-drop простором має чіткі обмеження на розмір та формат завантажуваних файлів (JPEG, PNG, MP4 до 100МБ) (Рисунок 3.24).

```
const SNIPPET_MAX = 50;
const STATEMENT_MIN = 100;
function countWords(str) {
  return str
    .trim()
    .split(/\s+/)
    .filter(Boolean)
    .length;
}
...
const MAX_MB = 100;
const MAX_BYTES = MAX_MB * 1024**2;
const tooBig = dragndropFiles.find(f => f.size > MAX_BYTES);
```

Рисунок 3.24 – Валідація введення та обмеження на файли

Результатом заповненого опенколу для митця/кині буде артворк зі статусом «РОЗГЛЯДАЄТЬСЯ». Для кураторів – ще одна заявка на розгляд.

3.7 Артворк

Рендер індексу LYR відбувається з масиву опублікованих артворків. Кожен індекс – айтем index-menu – це картка з порядковим номером, тайтлом артворку, митцем/кинею та датою публікації (Рисунок 3.25).

```
(async function buildIndexMenu () {
  const resp = await fetch('/api/artworks');
  ...
  artworks = await resp.json();

  indexMenu.innerHTML = '';

  artworks.forEach((art, i) => {
    const item = document.createElement('a');
    item.className = 'item';
    item.dataset.idx = i;
    if (i === 0) item.classList.add('item-1');
    ...
    item.innerHTML = `
      <span class="number">${i + 1}</span>
      <div class="item-info">
        <span class="project">${art.title.toUpperCase()},</span>
        <span class="artist-date">${art.artist_name.toUpperCase()},
${formatDate(art.published_at)}</span>
      </div>
      ${imgSrc ? `<img class="preview preview-img" src="${imgSrc}">` : ''}
      ${vidSrc ? `<video class="preview preview-vid" src="${vidSrc}" muted loop playsinline></
video>` : ''}
    `;
    ...
    indexMenu.appendChild(item);
  });
```

Рисунок 3.25 – Шаблон індексу артворку в index-menu

Кожен окремий артворк представлений модальним вікном openModal(), що накладається поверх індексу та надає повну взаємодію з конкретним артворком – його відкриття, закриття, перемикання між артворками (Рисунок 3.26).

```
async function openModal (idx) {
  currentIdx = idx;
  const art = artworks[idx];
  ...
  artworkHeader.innerHTML =
    `<span class="project">${art.title.toUpperCase()},</span><br>` +
    `<span class="artist-date">${art.artist_name.toUpperCase()},
${formatDate(art.published_at)}</span>`;

  statementBox.innerHTML = toParagraphs(art.statement || '');
  ...

  artworkModal.classList.remove('hidden');
}
```

Рисунок 3.26 – Відкриття модального вікна артворку

Оскільки медіафайли, порядок яких був заданий митцем/кинею в опенколі, зберігаються в uploads/artworks/, доступ до них здійснюється через media_paths. Модальне вікно артворку дозволяє перемикати медіа в межах окремого артворку в циклі (Рисунок 3.27).

```
setMedia(art.media_paths[0] || '');
artworkImg.onclick = artworkVid.onclick = () => {
  mediaIdx = (mediaIdx + 1) % art.media_paths.length;
  setMedia(art.media_paths[mediaIdx]);
};
```

Рисунок 3.27 – Перемикання медіа окремого артворку

Про резонанс авдиторії з артворком сигналізують «вподобання». Працюють вони через функцію like_artwork, яка додає новий рядок до таблиці artwork_likes та збільшує каунтер likes_count в таблиці artworks на 1. Якщо той самий користувач вже вподобав артворк, унікальний індекс спровокує unique_violation, а значить, дублювання не відбудеться (Рисунок 3.28).

```
router.post('/:id/like', async (req, res) => {
  const artId = +req.params.id;
  const ip = req.ip;
  ...
  try {
    ...
    if (exists.length) {
      ...
    } else {
      await pool.query(
        `SELECT like_artwork($1::integer, $2::inet, $3::integer)`,
        [artId, ip, user]
      );
    }
    ...
    res.json({ likes: rows[0].likes_count });
  } catch (err) {
    ...
  }
});
```

Рисунок 3.28 – Функція like_artwork

Навігація по артворках може здійснюватися не тільки за кліком на індекс, але й за допомогою кнопок ПОПЕРЕДНІЙ АРТВОРК та НАСТУПНИЙ АРТВОРК (Рисунок 3.29).

```
prevArtLabel.onclick = () => openModal(idx - 1);
nextArtLabel.onclick = () => openModal(idx + 1);
```

Рисунок 3.29 – Навігація prev/next

Основний акцент здійснюється на сам артворк, а тому було важливо не перевантажити модалку артворку зайвим функціоналом.

3.8 Мікровзаємодії та фінальний UI

Оскільки більшість артворків LYR існує у фото- та відеоформаті, виникла асоціація з фотооб'єктивом, що «фокусується» на кадрі. Звідси був створений кастомний курсор, що імітує вищеописане відчуття: вертикальна та горизонтальна лінії розміром 1px накладають на екран сітку, допомагаючи ніби «націлитися» на кожний елемент. (Рисунок 3.30).

```
window.addEventListener('mousemove', (e) => {
  if (rafId) return;
  rafId = requestAnimationFrame(() => {
    xLine.style.transform = `translateX(${e.clientX}px)`;
    yLine.style.transform = `translateY(${e.clientY}px)`;
    rafId = null;
  });
});
```

Рисунок 3.30 – Трансформація кастомного курсора

Аби кастомний курсор та фіксований хедер залишались видимими, для них був застосований стиль mix-blend-mode: difference, що інвертує колір тих пікселів, над якими вони опиняються: біла лінія на білому бекґраунді стає чорною; на чорному – білою; на кольоровому – дає контрастний «негатив», і нвпаки.

При проєктуванні UI, LYR свідомо експериментував з контрастом масштабів – з поєднанням багаторозмірних артворків, негативного простору та крихітного шрифту в одній композиції. Завдяки «паузам» створюється атмосфера неквапливого інтерфейсу, в якому розкриття артворку в повній мірі є пріоритетним завданням.

Так, на рисунках 3.31 та 3.32 представлено сторінки реєстрації та авторизації.

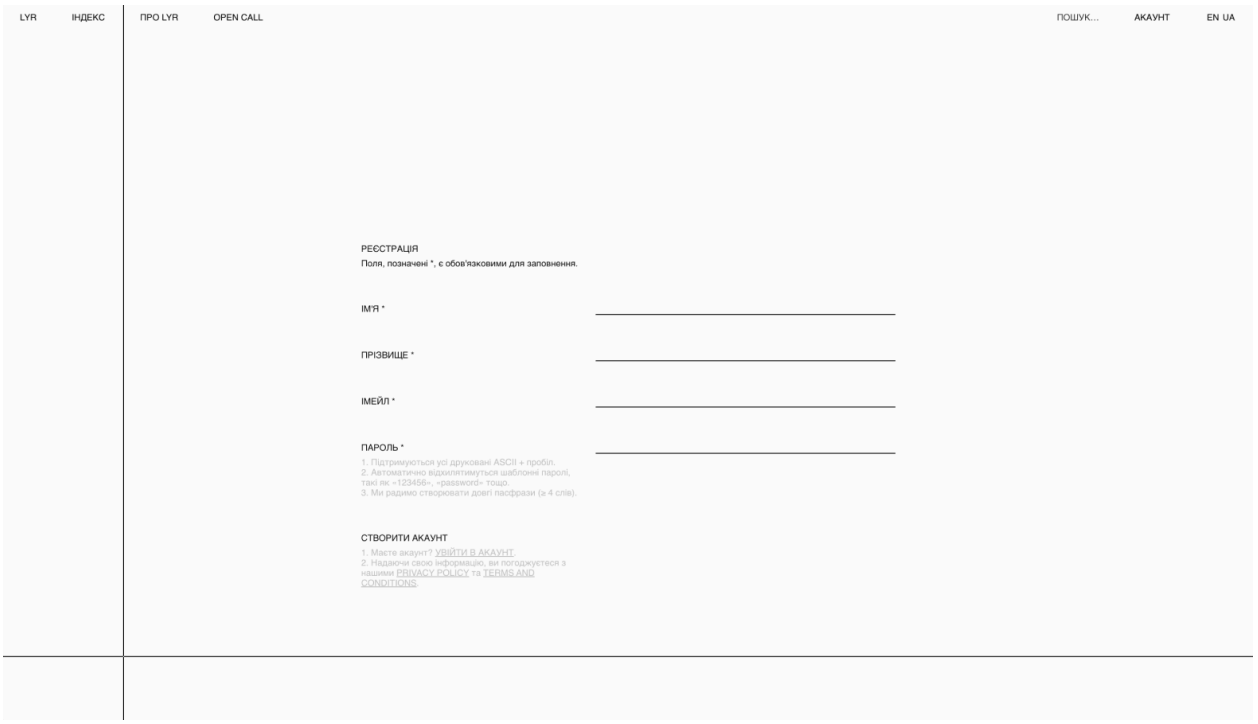

Рисунок 3.31 – Сторінка РЕЄСТРАЦІЯ

Рисунок 3.32 – Сторінка АВТОРИЗАЦІЯ

На рисунку 3.33 можна спостерігати сторінку з формою опенколу, тоді як на рисунку 3.34 – ПРО LYR.

Рисунок 3.33 – Сторінка OPEN CALL

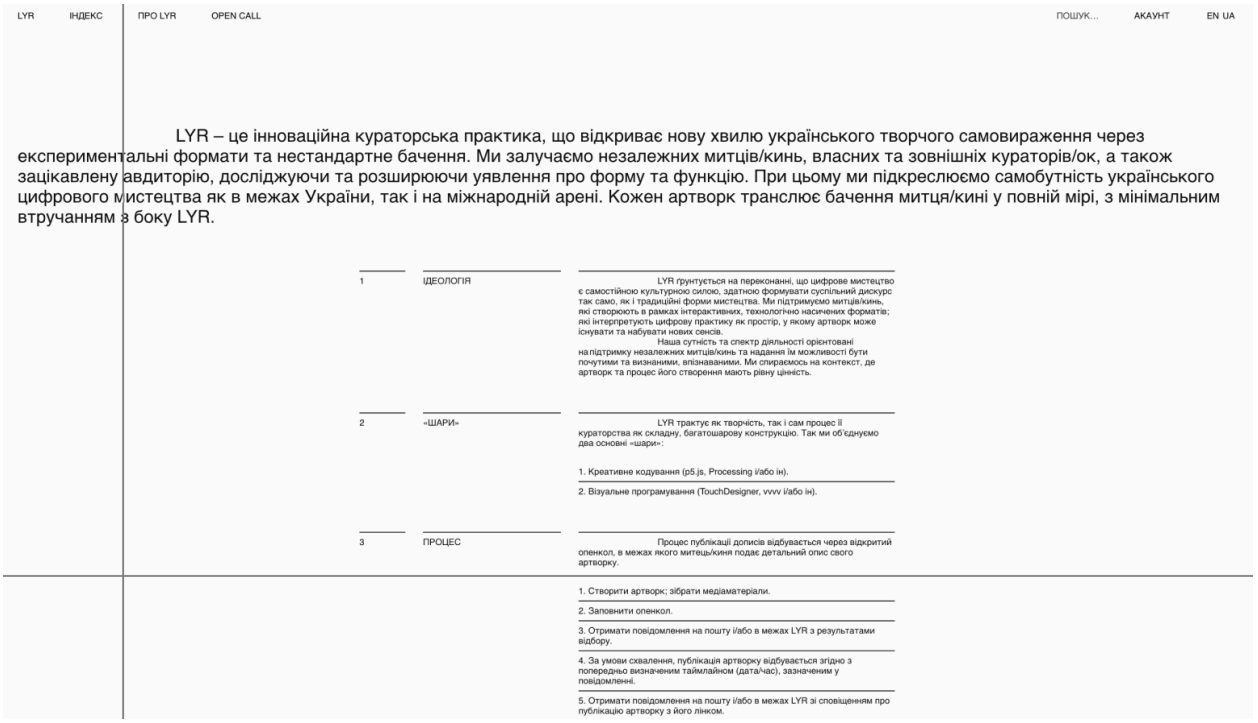

Рисунок 3.34 – Сторінка ПРО LYR

Щоб максимально перенести фокус користувача на модальному вікні, під час його відкриття на весь бекґраунд накладається фільтр розмиття –

blur(calc(var(--blur-base) / 2)). Розмиття накладається протягом всього часу, поки відкрита модалка; знімається при її закритті, створюючи м’який перехід фокусу з модалки знову на бекґраунд (Рисунок 3.35).

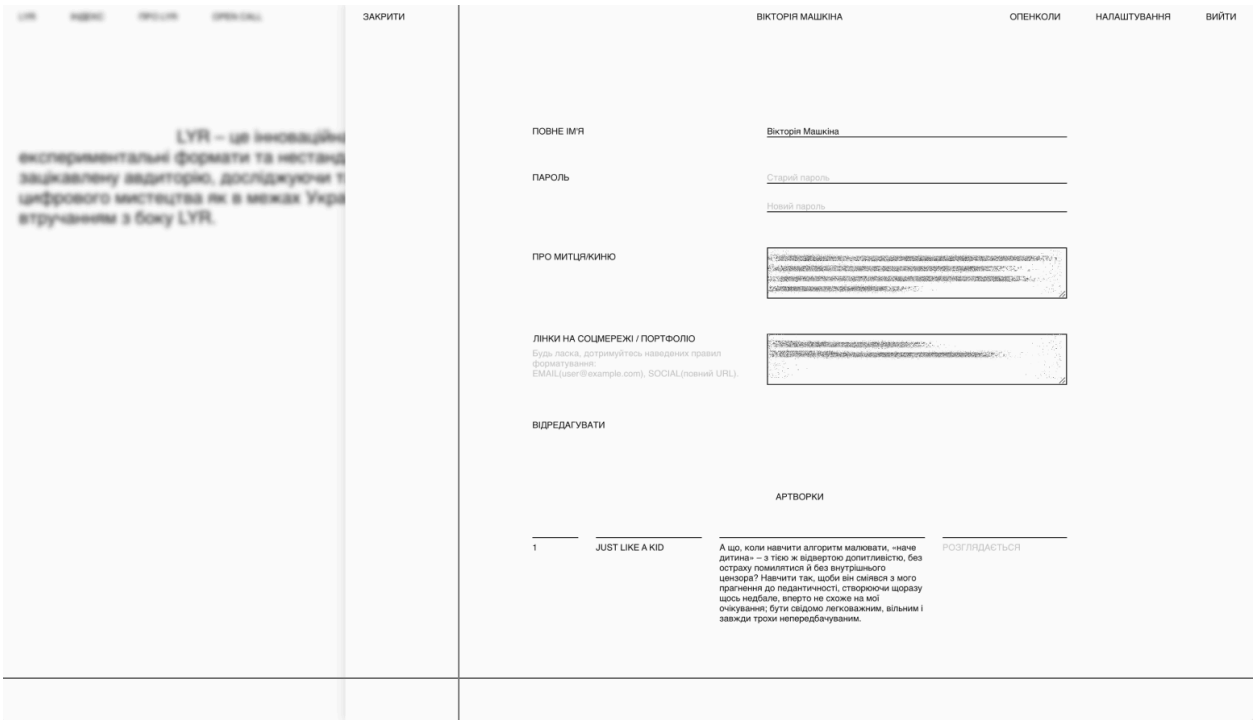

Рисунок 3.35 – Модальне вікно з активним НАЛАШТУВАННЯ

Активний при кліку рядок ґрід-списку ледве помітно затемнюється напівпрозорим сірим rgba(0, 0, 0, 0.05), щоб куратори LYR одразу бачили, з яким саме опенколом вони працюють (Рисунок 3.36).

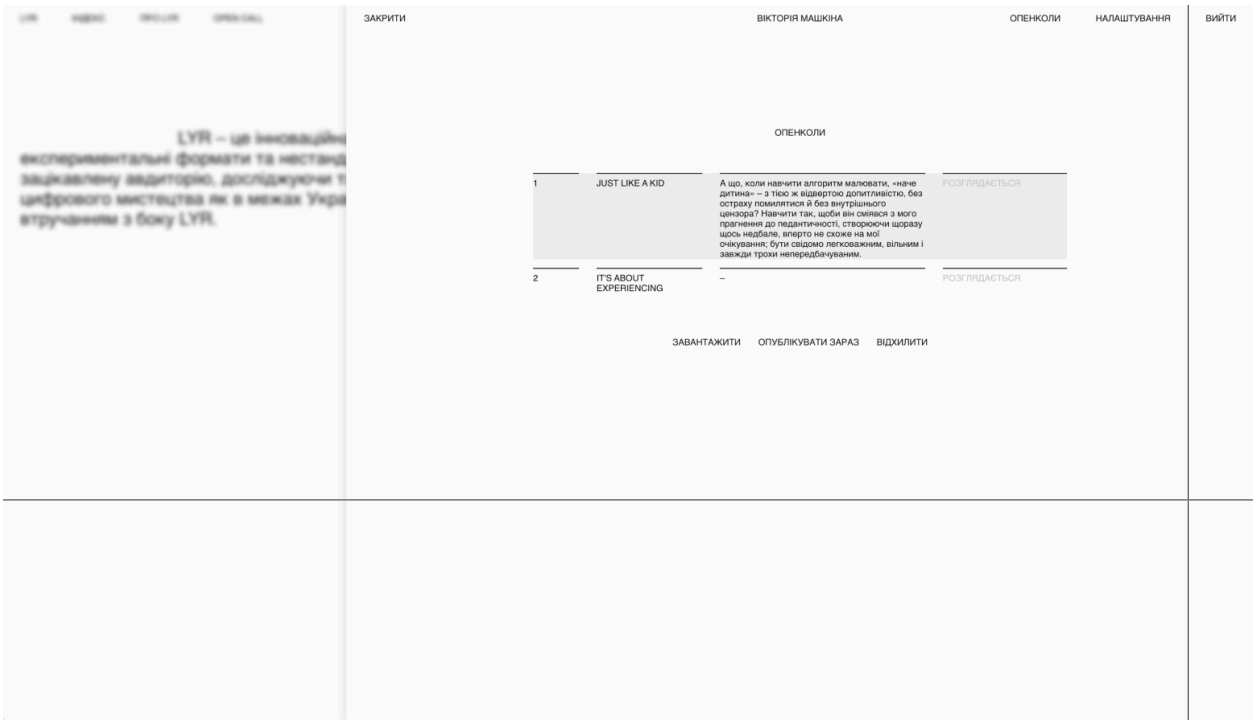

Рисунок 3.36 – Модальне вікно з активним ОПЕНКОЛИ

Модальне вікно АКАУНТ – те, що бачитиме будь-який користувач, коли захоче дізнатися більше про митця/киню та його/її творчість (Рисунок 3.37).

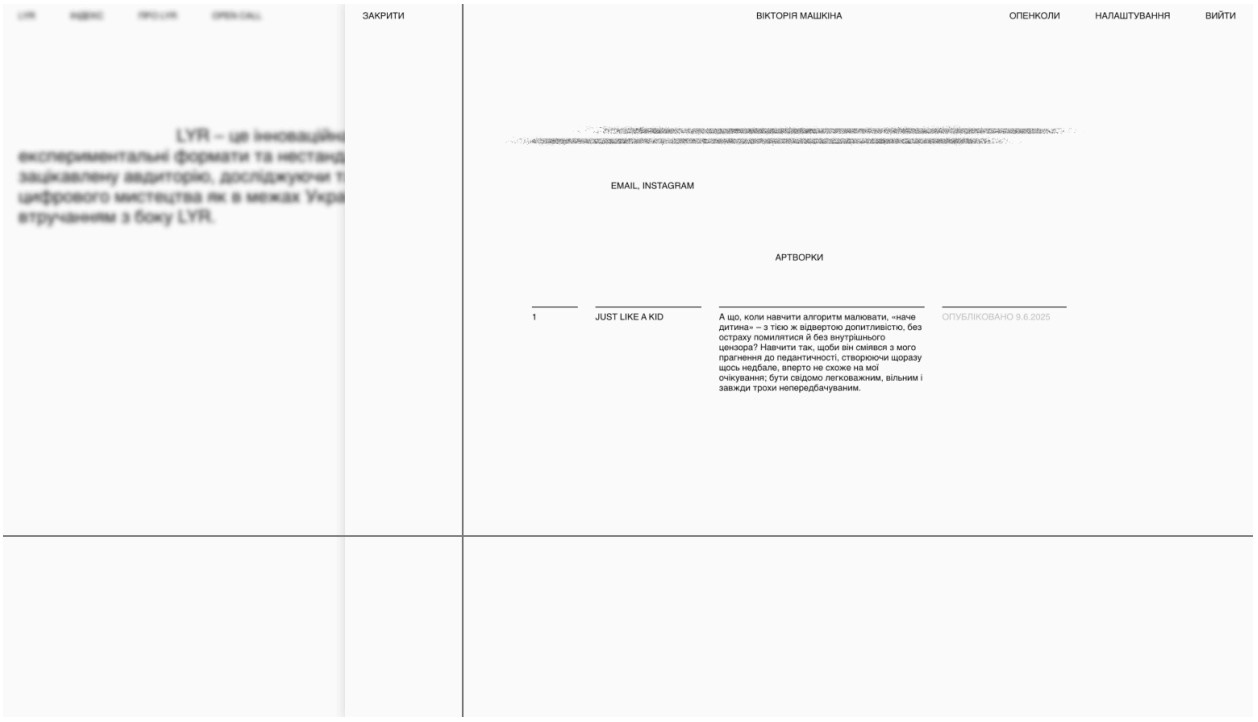

Рисунок 3.37 – Модальне вікно з активним АКАУНТ

Щоб користувач міг швидко переглянути будь-який артворк без відкриття модалки, реалізовано ховер-прев'ю: найперший доступний медіафайл – img або video – динамічно центрується та слідує за курсором в межах кожного айтему (Рисунок 3.38).

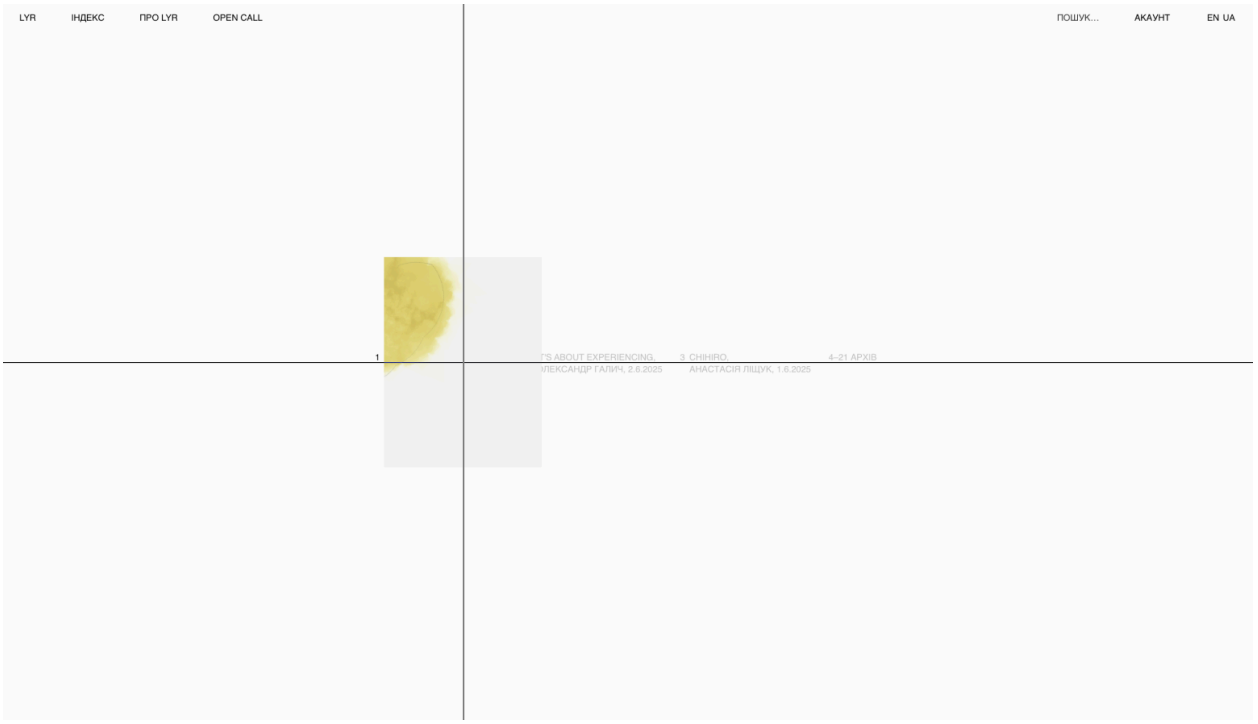

Рисунок 3.38 – Сторінка ІНДЕКС з активним ховер-прев’ю на першому індексі

За замовчуванням розгорнутий стейтмент артворку прихований. Тому щоб його відкрити, необхідно натиснути на «+» – текст накладеться на артворк, а бекґраунд під ним розмиється тим самим легким розмиттям (Рисунок 3.39).

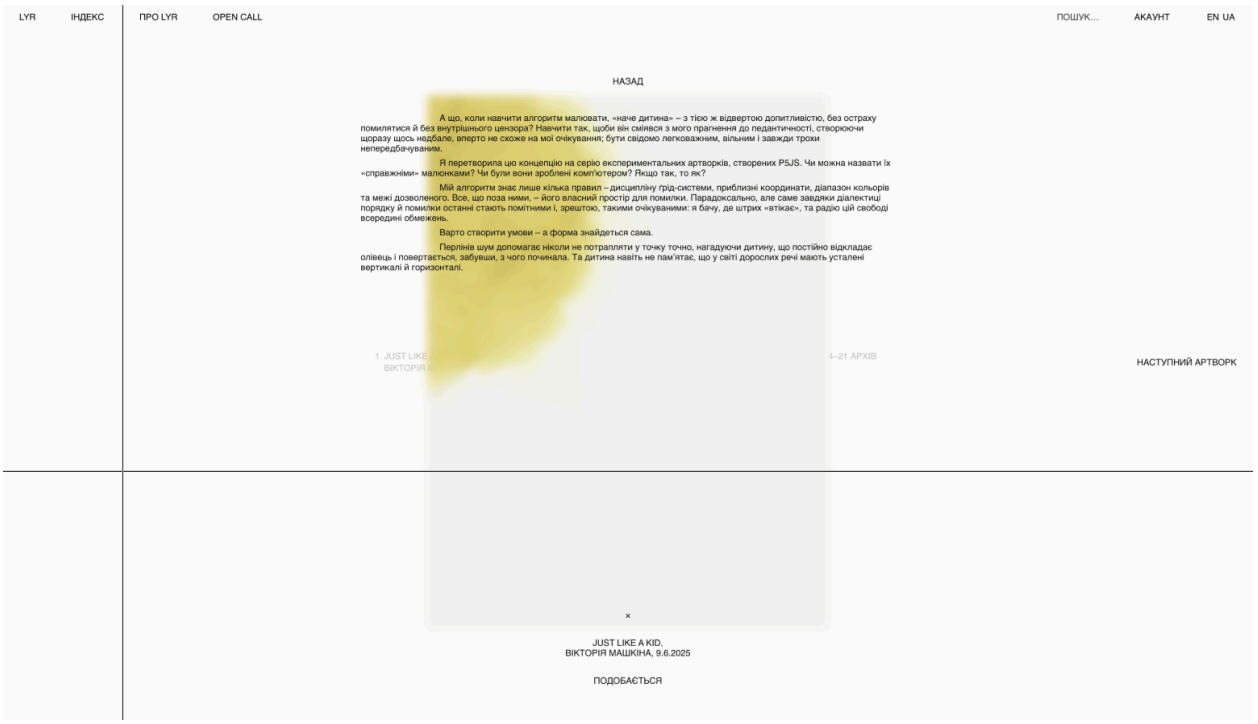

Рисунок 3.39 – Модальне вікно артворку з активним розгорнутим стейтментом

Крім того, для LYR було передбачено респонсивність (Рисунок 3.40).

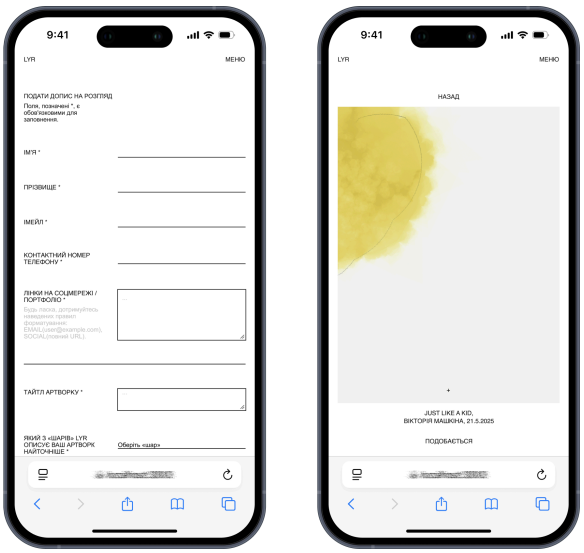

Рисунок 3.40 – Респонсивні екрани опенколу та артворку

Особиста інформація, що з’являлась на скріншотах інтерфейсу, була прихована з причини унеможливити зловживання персональними даними.

3.9 Тестування

В той час, як митці/кині шукають майданчик розголосу, куратори – стабільну, передбачувану інфраструктуру. Поєднати ці запити й одночасно застрахувати проєкт можна, зосередившись на тестуванні.

Пріоритетним сценарієм тестування був end-to-end (E2E) сценарій, що покриває повну ітерацію роботи користувача – як митця/кині, так і куратора – з об’єктом тестування – LYR. На момент E2E-тестування було створено окрему, ідентичну за структурою lyr_db базу даних lyr_testdb, яка обмежувалась лише обліковим записом адміністратора.

Відтак сценарій E2E-тестування наведений у таблиці 3.1.

Таблиця 3.1 – Сценарій E2E-тестування

№	Дія	Очікуваний результат
1	Митець/киня реєструється через /signup.html	Переадресація на /login.html; В БД: +1 запис в таблиці users, {is_admin=0}
2	Митець/киня авторизується через /login.html	Переадресація на /index.html, {id, is_admin=0}
3	Подається на опенкол через /open_call.html та заповнює форму	POST /api/artworks/submit повертає 201; В БД: +1 запис в таблиці artworks, title='TEST', status='РОЗГЛЯДАЄТЬСЯ'
4	ВИЙТИ в модалці	Переадресація на /login.html
5	Куратор авторизується через /login.html	Переадресація на /index.html; {id, is_admin=1}
6	ОПЕНКОЛИ в модалці	Секція view-pending активна
7	'TEST' ОПУБЛІКУВАТИ ЗАРАЗ	PATCH /api/artworks/:id/publish повертає 204; В БД: published_at=today(), status='ОПУБЛІКОВАНО', likes_count=0;

№	Дія	Очікуваний результат
		На фронті: index-menu +1 item
8	'TEST' ПОДОБАЄТЬСЯ/СКАСУВАТИ ВПОДОБАННЯ	likes_count +1/likes_count –1
9	Регрес статусу 'TEST'	В БД: published_at=NULL, status='РОЗГЛЯДАЄТЬСЯ'; На фронті: index-menu –1 item
10	'TEST' ВІДХИЛИТИ	PATCH /api/artworks/:id/reject повертає 204; В БД: status='ВІДХИЛЕНО', reject_note='TEST REJECT'

Сценарій тестувався кросбраузерно в Arc Browser (версія 1.97.1), Safari (версія 18.5) та Zen Browser (версія 1.12.10b).

Під час E2E-тестування було виявлено єдиний UI-дефект: якщо в модалці артворку розгорнути стейтмент, а потім перейти до попереднього/наступного артворку, бекґраунд під стейтментом залишався розмитим. Аби виправити це, на початку функції openModal() додано artworkModal.classList.remove('dimmed'), що скидає попередній стан перед рендером нового артворку та усуває небажане розмиття. На основну функціональність UI-дефект не вплинув – всі дії виконувалися коректно, а дані оновлювалися відповідно до очікуваних результатів.

Слід зазначити, що процес E2E-тестування фіксувався відеозаписом екрану за допомогою QuickTime Player (версія 10.5); результати зберігаються в docs/test/ із зазначенням дати проведення тестування.

ВИСНОВКИ

Цифрове мистецтво – і митці/кині, які його створюють, – «потрібне» з тих самих причин, з яких суспільство потребує будь-якого нового творчого медіуму: воно розширює можливості мистецтва, способи його сприйняття, а також історії та ідеї, які воно може висвітлювати.

LYR здатен передати це переконання через власне бачення цифрового мистецтва – як відкритого, технологічно насиченого, але водночас «спокійного» простору, в якому артворк та його митець/киня важать однаково.

Простір LYR було задумано як платформу для експонування двох практик – креативного кодування та візуального програмування – із моделлю відкритого опенколу. Було проведено аналіз потенційного таргету LYR, а також сучасних технологій, аби максимально потрапляти в інтереси та очікування цільової авдиторії.

Наслідком було розроблено вебплатформу LYR, яка об’єднує мистецькі траєкторії незалежних митців/кинь, внутрішніх та зовнішніх кураторів, а також зацікавлену авдиторію в одній точці.

У довгостроковій перспективі LYR слугуватиме українським архівним майданчиком експериментальних цифрових форматів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Artsy – Discover and Buy Fine Art. Artsy. URL: https://www.artsy.net.
2. Spilne Art – українська мистецька платформа. Spilne Art. URL: https://spilne.art/.
3. Behance – For You :: Behance. Behance. URL: https://www.behance.net/for_you.
4. Favicon – Glossary | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Glossary/Favicon.
5. HTML: HyperText Markup Language | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/HTML.
6. CSS: Cascading Style Sheets | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/CSS.
7. JavaScript | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript.
8. Express/Node introduction – Learn web development | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/Express_Nodejs/Introduction.
9. Node.js – An introduction to the npm package manager. Node.js – Run JavaScript Everywhere. URL: https://nodejs.org/en/learn/getting-started/an-introduction-to-the-npm-package-manager.
10. Technology | 2024 Stack Overflow Developer Survey. Stack Overflow Insights - Developer Hiring, Marketing, and User Research. URL: https://survey.stackoverflow.co/2024/technology.
11. PostgreSQL. PostgreSQL. URL: https://www.postgresql.org/.
12. DBngin | All-in-One Database Version Management Tool. DBngin | All-in-One Database Version Management Tool. URL: https://dbngin.com/.

13. TablePlus | Modern, Native Tool for Database Management. TablePlus | Modern, Native Tool for Database Management. URL: https://tableplus.com/.
14. Microsoft. Visual Studio Code – Code Editing. Redefined. Visual Studio Code – Code Editing. Redefined. URL: https://code.visualstudio.com/.
15. Version control – Learn web development | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Version_control.

Додаток А – Код програми

Файл «signup.js»

```
const form = document.getElementById('signup-form');
const passInput =
document.getElementById('signup-password');

const blackList = [
  '123456', 'password', 'qwerty', '111111', 'letmein',
  '12345678', '123456789', 'abc123', 'admin', 'welcome'
];

const PASS_MIN_LENGTH = 6;
const PASS_MIN_WORDS = 4;

form.addEventListener('submit', async (e) => {
  e.preventDefault();

  const pass = passInput.value;
  if (!/^[\x20-\x7E]+$/.test(pass)) {
    alert('Пароль може містити лише друковані ASCII-символи
й пробіли.');
    return;
  }
  if (pass.length < PASS_MIN_LENGTH) {
    alert(`Пароль має містити щонайменше ${PASS_MIN_LENGTH}
символів.`);
    return;
  }
  if (blackList.includes(pass.toLowerCase())) {
    alert('Пароль занадто простий.');
    return;
  }
  const wordCount =
pass.trim().split(/\s+/).filter(Boolean).length;
  if (wordCount < PASS_MIN_WORDS) {
    const goOn = confirm(
      `Ваш пароль складається з ${wordCount} слова/ів.\n` +
      `Рекомендуємо використовувати довгі пасфрази (≥
${PASS_MIN_WORDS} слів).\n` +
      'Продовжити реєстрацію з поточним паролем?'
    );
    if (!goOn) return;
  }

  const fd = new FormData(form);
  const body = Object.fromEntries(fd.entries());

  try {
    const r = await fetch('/api/users/register', {
      method : 'POST',
```

```
          headers: { 'Content-Type': 'application/json' },
          body   : JSON.stringify(body)
        });

        if (r.status === 201) {
          alert('Реєстрація успішна — увійдіть.');
          location.href = '../login.html';
        } else {
          const { message } = await r.json();
          alert(message || 'Помилка реєстрації');
        }
      } catch {
        alert('Сервер недоступний');
      }
    });
```

Файл «login.js»

```
    const form = document.getElementById('login-form');
    const passInput =
document.getElementById('login-password');

    form.addEventListener('submit', async (e) => {
      e.preventDefault();

      if (!/^[\x20-\x7E]+$/.test(passInput.value)) {
        alert('Пароль може містити лише друковані ASCII-символи
й пробіли.');
        return;
      }

      const fd = new FormData(form);
      const body = Object.fromEntries(fd.entries());

      try {
        const r = await fetch('/api/users/login', {
          method: 'POST',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify(body)
        });

        if (r.ok) {
          const { id, is_admin } = await r.json();
          localStorage.setItem('userId',  id);
          localStorage.setItem('isAdmin', is_admin);
          location.href = '../index.html';
        } else {
          const { message } = await r.json().catch(() => ({}));
          alert(message || 'Невірні облікові дані.');
        }
      } catch {
        alert('Сервер недоступний');
```

```
  }
});
```

Файл «open-call-submit.js»

```
const uid  = localStorage.getItem('userId');
const form = document.getElementById('open-call-form');
const fileInput = document.getElementById('media');

function checkLogin(e) {
  if (!uid) {
    e && e.preventDefault();
    location.href = '../login.html';
    return true;
  }
  return false;
}
form.addEventListener('focusin', checkLogin, { once: true
});

const SNIPPET_MAX = 50;
const STATEMENT_MIN = 100;
const MAX_MB = 100;
const MAX_BYTES = MAX_MB * 1024**2;

function countWords(str) {
  return str
    .trim()
    .split(/\s+/)
    .filter(Boolean)
    .length;
}

function updateCounter(el, cntEl, limit, isUpperBound) {
  const words = countWords(el.value);
  cntEl.textContent = isUpperBound
    ? `${words} / ${limit}`
    : `${words} / ${limit}+`;

  const invalid = isUpperBound
    ? words > limit
    : words < limit;

  cntEl.classList.toggle('invalid', invalid);
}

const snippetElement  = document.getElementById('snippet');
const snippetCounter =
document.getElementById('snippet-counter');
const statementElement =
document.getElementById('statement');
```

```
    const statementCounter =
document.getElementById('statement-counter');

    updateCounter(snippetElement, snippetCounter, SNIPPET_MAX,
true);
    updateCounter(statementElement, statementCounter,
STATEMENT_MIN, false);

    snippetElement.addEventListener('input', () =>
      updateCounter(snippetElement, snippetCounter,
SNIPPET_MAX, true)
    );
    statementElement.addEventListener('input', () =>
      updateCounter(statementElement, statementCounter,
STATEMENT_MIN, false)
    );

    form.addEventListener('submit', async (e) => {
      if (checkLogin(e)) return;
      e.preventDefault();

      const snippetWords = countWords(snippetElement.value);
      const statementWords =
countWords(statementElement.value);

      if (snippetWords > SNIPPET_MAX) {
        alert(`Сніпет має містити не більше ${SNIPPET_MAX}
слів.`);
        return;
      }
      if (statementWords < STATEMENT_MIN) {
        alert(`Стейтмент має містити щонайменше
${STATEMENT_MIN} слів.`);
        return;
      }

      const dragndropFiles = window.selectedFiles?.length
        ? window.selectedFiles
        : Array.from(fileInput.dragndropFiles);

      const tooBig = dragndropFiles.find(f => f.size >
MAX_BYTES);
      if (tooBig) {
        alert(`Файл «${tooBig.name}» перевищує ліміт у
${MAX_MB} МБ.`);
        return;
      }

      const fd = new FormData(form);
      fd.append('user_id', uid);

      const files = window.selectedFiles?.length
        ? window.selectedFiles
```

```
          : Array.from(fileInput.files);
        files.forEach(f => fd.append('media[]', f));

        try {
          const r = await fetch('/api/artworks/submit', { method:
'POST', body: fd });
          if (r.status === 201) {
            location.href = '../index.html';
          } else {
            const { message } = await r.json().catch(() => ({}));
            alert(message || `Сервер відповів ${r.status}`);
          }
        } catch (err) {
          console.error(err);
          alert('Сервер недоступний');
        }
      });
```

Файл «index-load.js»

```
      const indexMenu = document.getElementById('index-menu');
      const artworkModal =
document.getElementById('artwork-modal');

      const artworkImg =
document.getElementById('artwork-image');
      const artworkVid =
document.getElementById('artwork-video');

      const statementBox = document.getElementById('more-text');
      const artworkHeader =
document.getElementById('artwork-header');

      const likeButton = document.getElementById('like-button');
      const likeCounter = document.getElementById('like-count');

      const prevArtLabel = document.getElementById('prev-label');
      const nextArtLabel = document.getElementById('next-label');

      const toggleMoreButton =
document.getElementById('toggle-button');
      const backButton = document.getElementById('back-label');

      const previewOverlayVideo =
document.querySelector('.video-overlay');

      const contactsBox = document.createElement('div');
      contactsBox.className = 'user-contacts';

      let artworks = [];
      let currentIdx = -1;
```

```
    const isVideo = src => /\.mp4$/i.test(src);


    (async function buildIndexMenu () {
      const resp = await fetch('/api/artworks');
      if (!resp.ok) return;
      artworks = await resp.json();

      indexMenu.innerHTML = '';

      artworks.forEach((art, i) => {
        const item = document.createElement('a');
        item.className = 'item';
        item.dataset.idx = i;
        if (i === 0) item.classList.add('item-1');

        const imgSrc = art.media_paths.find(src =>
!isVideo(src));
        const vidSrc = art.media_paths.find(src =>
isVideo(src));

        item.innerHTML = `
          <span class="number">${i + 1}</span>
          <div class="item-info">
            <span
class="project">${art.title.toUpperCase()},</span>
            <span
class="artist-date">${art.artist_name.toUpperCase()},
${formatDate(art.published_at)}</span>
          </div>
          ${imgSrc ? `<img class="preview preview-img"
src="${imgSrc}">` : ''}
          ${vidSrc ? `<video class="preview preview-vid"
src="${vidSrc}" muted loop playsinline></video>` : ''}
        `;

        item.addEventListener('click', e => {
e.preventDefault(); openModal(i); });

        const imgPrev = item.querySelector('.preview-img');
        const vidPrev = item.querySelector('.preview-vid');

        item.addEventListener('mouseenter', () => {
          if (vidPrev) { vidPrev.style.display = 'block';
vidPrev.play(); }
          else if (imgPrev) imgPrev.style.display = 'block';
        });
        item.addEventListener('mouseleave', () => {
          if (vidPrev) { vidPrev.pause(); vidPrev.style.display
= 'none'; }
          else if (imgPrev) imgPrev.style.display = 'none';
        });
```

```
        item.addEventListener('mousemove', e => {
          const prevEl = vidPrev?.style.display === 'block' ?
vidPrev
                       : imgPrev?.style.display === 'block' ?
imgPrev
                       : null;
          if (!prevEl) return;

          const rect = item.getBoundingClientRect();
          const relX = e.clientX - rect.left;
          const relY = e.clientY - rect.top;

          const maxLeft = window.innerWidth -
prevEl.offsetWidth - rect.left;
          const maxTop = window.innerHeight -
prevEl.offsetHeight - rect.top;

          let left = relX - prevEl.offsetWidth / 2;
          let top = relY - prevEl.offsetHeight / 2;

          if (left < -rect.left) left = -rect.left;
          if (left > maxLeft) left = maxLeft;
          if (top < -rect.top) top = -rect.top;
          if (top > maxTop) top = maxTop;

          prevEl.style.transform = `translate(${left}px,
${top}px)`;
        });

        indexMenu.appendChild(item);
      });

      const archive = document.createElement('div');
      archive.className = 'item';
      archive.innerHTML = `
        <span class="number">${artworks.length +
1}-21 APXIB</span>
        <div class="item-info"><span
class="project"></span><span class="artist-date"></span></div>`;
      indexMenu.appendChild(archive);

      const firstVid = artworks[0]?.media_paths.find(isVideo);
      if (firstVid) previewOverlayVideo.src = firstVid;
    })();


    async function openModal (idx) {
      currentIdx = idx;
      const art = artworks[idx];
      const userId = localStorage.getItem('userId');

      artworkModal.classList.remove('dimmed');
```

```
        indexMenu.style.color = 'var(--color-silver-sand)';

        artworkHeader.innerHTML =
          `<span
class="project">${art.title.toUpperCase()},</span><br>` +
          `<span
class="artist-date">${art.artist_name.toUpperCase()},
${formatDate(art.published_at)}</span>`;

        statementBox.innerHTML = toParagraphs(art.statement ||
'');

        const detail = await
fetch(`/api/artworks/${art.id}`).then(r => r.json());
        contactsBox.innerHTML = fmtContacts(detail.contacts ||
'');
        statementBox.appendChild(contactsBox);

        let mediaIdx = 0;
        const setMedia = src => {
          const v = isVideo(src);
          artworkImg.src = v ? '' : src;
          artworkVid.src = v ? src : '';
          artworkImg.classList.toggle('active-media', !v);
          artworkImg.classList.toggle('inactive-media', v);
          artworkVid.classList.toggle('active-media', v);
          artworkVid.classList.toggle('inactive-media', !v);
          if (v) artworkVid.play(); else artworkVid.pause();
        };
        setMedia(art.media_paths[0] || '');
        artworkImg.onclick = artworkVid.onclick = () => {
          mediaIdx = (mediaIdx + 1) % art.media_paths.length;
          setMedia(art.media_paths[mediaIdx]);
        };

        await loadLikes(art.id);
        likeButton.onclick = async () => {
          const r = await fetch(`/api/artworks/${art.id}/like`, {
            method: 'POST',
            headers: { 'X-User-Id': userId }
          });
          if (r.ok) {
            const { likes } = await r.json();
            likeCounter.textContent = likes;
            const { liked } = await
fetch(`/api/artworks/${art.id}/likes`, {
              headers: { 'X-User-Id': userId }
            }).then(r => r.json());
            setLikeButton(liked);
          }
        };

        prevArtLabel.style.display = idx ? '' : 'none';
```

```
      nextArtLabel.style.display = idx < artworks.length - 1 ?
'' : 'none';
      prevArtLabel.textContent = 'ПОПЕРЕДНІЙ АРТВОРК';
      nextArtLabel.textContent = 'НАСТУПНИЙ АРТВОРК';
      prevArtLabel.onclick = () => openModal(idx - 1);
      nextArtLabel.onclick = () => openModal(idx + 1);

      let shown = false;
      toggleMoreButton.textContent = '+';
      statementBox.classList.add('hidden');
      toggleMoreButton.onclick = () => {
        shown = !shown;
        toggleMoreButton.textContent = shown ? '×' : '+';
        statementBox.classList.toggle('hidden', !shown);
        artworkModal.classList.toggle('dimmed', shown);
      };

      artworkModal.classList.remove('hidden');
    }

    backButton.onclick = () => {
      artworkModal.classList.add('hidden');
      prevArtLabel.style.display = 'none';
      nextArtLabel.style.display = 'none';
      indexMenu.style.color = '';
    };


    async function loadLikes (artId) {
      const userId = localStorage.getItem('userId');
      const r = await fetch(`/api/artworks/${artId}/likes`, {
        headers: { 'X-User-Id': userId }
      });
      if (!r.ok) return;
      const { likes, liked } = await r.json();
      likeCounter.textContent = likes;
      setLikeButton(liked);
    }

    function setLikeButton (liked) {
      if (liked) {
        likeButton.innerHTML =
'ВПОДОБАНО!<br>СКАСУВАТИ ВПОДОБАННЯ';
      } else {
        likeButton.textContent = 'ПОДОБАЄТЬСЯ';
      }
    }

    function formatDate (iso) {
      const d = new Date(iso);
      return `${d.getDate()}.${d.getMonth() +
1}.${d.getFullYear()}`;
    }
```

```
      function fmtContacts (raw) {
        return raw.split(',')
          .map(t => t.trim())
          .map(t => {
            const m = t.match(/^([^(]+)\(([^)]+)\)$/);
            if (!m) return t;
            const label = m[1].trim();
            const url = m[2].trim();
            const href = /@/.test(url) ? `mailto:${url}` : url;
            return `<a href="${href}"
target="_blank">${label}</a>`;
          }).join(', ');
      }

      function toParagraphs (txt) {
        return txt
          .split(/(?:<br\s*\/?>|\n)+/i)
          .map(s => s.trim())
          .filter(Boolean)
          .map(p => `<p>${p}</p>`)
          .join('');
      }
```