

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, назва факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавра

(ступінь вищої освіти)

на тему РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ПОЕТАПНОГО
ГРОМАДСЬКОГО ФІНАНСУВАННЯ МУЗИЧНИХ ПРОЄКТІВ
SOFTWARE DEVELOPMENT FOR GRADUAL CROWDFUNDING OF MUSIC
PROJECTS

Виконав: студент(ка) 4 курсу, групи КНТз-117
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Маковський Я.Р.

(прізвище та ініціали)

Керівник Неласа Г.В.

(прізвище та ініціали)

Рецензент Шитікова О.В.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ІРЕ, ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
“ ” 2021 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Маковського Ярослава Романовича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка програмного забезпечення поетапного
громадського фінансування музичних проєктів
Software Development for Gradual Crowdfunding of Music Projects

керівник проєкту (роботи) Неласа Ганна Вікторівна, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “30” березня 2021 року № 103

2. Строк подання студентом проєкту (роботи) 17.05.2021

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне
завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми поетапного громадського фінансування музичних проєктів. 2. Розробка архітектури програми. 3. Розробка програми. 4. Експлуатація та тестування програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Неласа Г.В., професор		
Нормоконтролер	Андрєєв М.О., асистент		

7. Дата видачі завдання “12” березня 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2-3 тижні	Розділ 1
3	Розробка архітектури програми	4 тиждень	Розділ 2
4	Розробка програми	5-6 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми	7 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	8 тиждень	Додатки
7	Захист роботи	9 тиждень	

Студент(ка)

(підпис) Маковський Я.Р.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис) Неласа Г.В.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 80 с., 29 рис., 2 табл., 3 дод., 16 джерел.

МУЗИЧНИЙ ПРОЄКТ, ГРОМАДСЬКЕ ФІНАНСУВАННЯ, ВЕБЗАСТОСУНОК, ОБ'ЄКТНО-РЕЛЯЦІЙНЕ ВІДОБРАЖЕННЯ, DJANGO, PYTHON.

Об'єкт роботи – процес поетапного громадського фінансування музичних проєктів. Предмет роботи – програмне забезпечення поетапного громадського фінансування музичних проєктів.

Мета роботи – розроблення програмного забезпечення поетапного громадського фінансування музичних проєктів для підтримки безпечної процедури фінансування.

Матеріали, методи та технічні засоби: мова програмування Python, вебфреймворк Django, система керування базами даних MySQL.

Результати. Проєктування і реалізацію програмного забезпечення виконано за результатами аналізу предметної області з моделюванням роботи користувачів з програмою, визначенням інструментів розробки, архітектури програмного забезпечення, проєктуванням структури даних, описом розроблених класів.

Висновки. Вебзастосунок, який було розроблено в роботі, надає можливість створення заявок на фінансування музичних проєктів, керування цими заявками, створення власних внесків на фінансування музичних проєктів, які надаються проєкту поетапно, після того, як куратор проєкту підтвердив виконання попереднього етапу, і повертаються користувачу, якщо потрібну суму на фінансування не вдалося зібрати.

Галузь використання – пошук коштів на фінансування музичних проєктів.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 80 pages, 29 figures, 2 tables, 3 appendixes, 16 sources.

MUSIC PROJECT, GRADUAL CROWDFUNDING, WEB APPLICATION, OBJECT-RELATIONAL MAPPING, DJANGO, PYTHON.

The object of work is the process of gradual public financing of music projects. The subject of the work is the software of stage-by-stage public financing of music projects.

The purpose of the work is to develop software for gradual crowdfunding of music projects to support a secure funding procedure.

Materials, methods and technical means: Python programming language, Django web framework, MySQL database management system.

Results. Design and implementation of software is performed based on the analysis of the subject area with modeling of users' work with the program, definition of development tools, software architecture, design of data structure, description of developed classes.

Conclusions. The web application, which was developed in the work, provides the ability to create applications for funding music projects, manage these applications, create own contributions to finance music projects, which are provided to the project in stages, after the project curator confirmed the previous stage, and are returned to users if the required amount of funding could not be raised.

Field of use is search for funds to finance music projects.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	7
Вступ.....	8
1 Аналіз проблеми поетапного громадського фінансування музичних проєктів	9
1.1 Проблема громадського фінансування музичних проєктів.....	9
1.2 Існуючі програмні сервіси громадського фінансування музичних проєктів.....	11
1.3 Вимоги до програмного забезпечення	17
1.4 Висновки за розділом 1	18
2 Розробка архітектури програми.....	19
2.1 Вибір технології розробки.....	19
2.2 Архітектура програмного забезпечення	20
2.3 Функціональні вимоги до програмного забезпечення	22
2.4 Проєктування структури даних	26
2.5 Висновки за розділом 2	30
3 Розробка програми	32
3.1 Опис послідовності роботи програми.....	32
3.2 Опис реалізації програми	33
3.3 Висновки за розділом 3	41
4 Експлуатація та тестування програми	42
4.1 Призначення програми	42
4.2 Умови виконання програми	42
4.3 Робота з програмою	42
Висновки	50
Перелік джерел посилання	51
Додаток А Технічне завдання	53
Додаток Б Текст програми	57
Додаток В Слайди презентації.....	73

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

DTL – Django Template Language;

MVT – Model-View-Template.

ВСТУП

Музиканти завжди потребували коштів на реалізацію своєї діяльності. Раніше такі кошти могли надати їм продюсери або підприємства, які брали участь у такому продюсуванні і фінансуванні. На даний момент з'явилися спеціалізовані платформи, які дозволяють проєктам з різних галузей забезпечувати громадське фінансування. У такому випадку кошти для реалізації проєктів надають не великі спонсори, а маленькі, в якості яких можуть виступати звичайні люди. З маленьких сум індивідуальних внесків формується потрібна загальна сума. При цьому звичайно, що спонсори впливають на роботу музикантів, але при цьому вони отримують більший ступінь незалежності порівняно з фінансуванням великих власників, які відповідно найчастіше починають диктувати стратегію розвитку проєкту, а також впливати і на контент.

У результаті платформи громадського фінансування проєктів можна розглядати як інструмент забезпечення більшого рівня незалежності для музикантів, які бажають реалізувати власні ідеї. На даний момент існують великі платформи, які забезпечують підтримку даного процесу. Проте варто враховувати, що часто існують окремі спільноти людей, які потребують створення власного центру комунікації, окрім того платформа може враховувати регіональні особливості, у такому випадку як раз і виникає потреба у створенні програмного забезпечення громадського фінансування музичних проєктів. Окрім того сама галузь розваг є достатньо популярною і великою у світі, а тому характеризується великими коштами, які потребують різних інструментів для створення відповідної екосистеми.

Метою роботи було обрано розроблення програмного забезпечення поетапного громадського фінансування музичних проєктів для забезпечення процесу збору коштів, захищеного за рахунок використання створеної платформи в ролі посередника.

1 АНАЛІЗ ПРОБЛЕМИ ПОЕТАПНОГО ГРОМАДСЬКОГО ФІНАНСУВАННЯ МУЗИЧНИХ ПРОЄКТІВ

1.1 Проблема громадського фінансування музичних проєктів

Громадське фінансування музичних проєктів означає отримання невеликої кількості підтримки з боку великої аудиторії, це дозволяє перестати шукати потрібний для внесення аванс з одного джерела (наприклад, звукозаписна компанія чи позика) [1].

Музичні виконавці дедалі частіше звертаються до своєї аудиторії за фінансуванням, переконуючи шанувальників внести свій внесок у випуск альбому, продаж одноразових продуктів шанувальникам – це звична практика, що може включати виступ в прямому ефірі для фанатів, написання піснi на замовлення, виступ у деякому місті, платформи для громадського фінансування музичних проєктів формалізують даний процес фінансування: через вебсайт виконавця або групи та через присутність у соціальних мережах шанувальників просять внести свій внесок у проєкт, музиканти при цьому повністю зберігають свої права, а вебсайт стягує комісію, як тільки буде узгоджена ціль, все це потребує додаткових дій, зокрема створення своєї бази фанатів в Інтернеті [2].

Важливо, щоб в результаті такі вебсайти зберігали кошти на депозитному рахунку, тоді можна говорити про те, що громадське фінансування музичних проєктів – це ефективний спосіб випуску альбомів, оскільки він дозволяє артистам і групам перевірити, чи є достатній попит на їх музику, перш ніж вони зобов'язуються випустити її, при цьому якщо достатньо шанувальників зроблять свої внески, то проєкт досягає своєї мети, а музикант записує, виготовляє та розповсюджує альбом у заздалегідь визначені терміни, якщо мета не досягнута, музиканти не зобов'язані створювати альбом [2].

Перш ніж розпочати свою кампанію, слід визначити і дотримуватися наступних принципів:

- потрібно визначити мету і знати, що потрібно для її досягнення, бути реалістичним, так мета повинна бути репрезентативною для рекламного впливу, для цього потрібно спочатку виконати дослідження стану ринку;

- визначити терміни: деякі сайти подібного типу вимагають встановити термін, це важливо з ряду причин: ця властивість важлива з декількох причин, адже по-перше, не можна використовувати гроші до закінчення терміну збору, більше того, найчастіше потрібно насправді успішно досягти встановлену цифру протягом визначеного часу;

- звичайно зусилля зі збору коштів не будуть серйозними, якщо музикант не поділиться інформацією про збір коштів якомога більше, потрібно постійно рекламувати свою кампанію на будь-якому каналі соціальних мереж, потрібно використовувати посилання на кампанію в YouTube і Soundcloud;

- щоб вдало репрезентувати проєкт потрібно розповісти свою історію: важливо бути максимально особистими, люди не збираються вкладати гроші, якщо не знають, куди вони йдуть, незалежно від того, чи записується відео для презентації, чи пишеться стаття, обов'язково потрібно розповісти свою історію, адже люди вкладають у людей;

- потрібно пояснити, для чого саме потрібні гроші, розповісти про своє бачення і, перш за все, бути абсолютно прозорими щодо проєкту;

- потрібно подякувати за кожен внесок: це важлива частина будь-якої кампанії, люди вкладають гроші вперед, щоб допомогти досягти мрії автору проєкту, це справжні шанувальники, які піклуються про успіх автора, тому потрібно обов'язково подякувати, а це можна зробити різними способами – більшість платформ дозволяє створити спеціальний електронний лист із вдячністю, потрібно постійно інформувати людей про прогрес;

- непоганим інструментом є запропонувати ряд рівнів винагород, щоб стимулювати більші пожертви: чим більше грошей вносить людина, тим більшою буде їй віддача;

– якщо кампанія не працює, потрібно бути готовим змінити її та спробувати щось нове [3].

1.2 Існуючі програмні сервіси громадського фінансування музичних проєктів

Kickstarter [4] (рис. 1.1) є одним з найвідоміших засобів фінансування проєктів у сфері культури. Функціонує за принципом «все або нічого», тобто якщо проєкт протягом заданого строку не зміг отримати ту суму коштів, яка була його ціллю, то всі кошти, зібрані за проєктом, повертаються спонсорам [5].

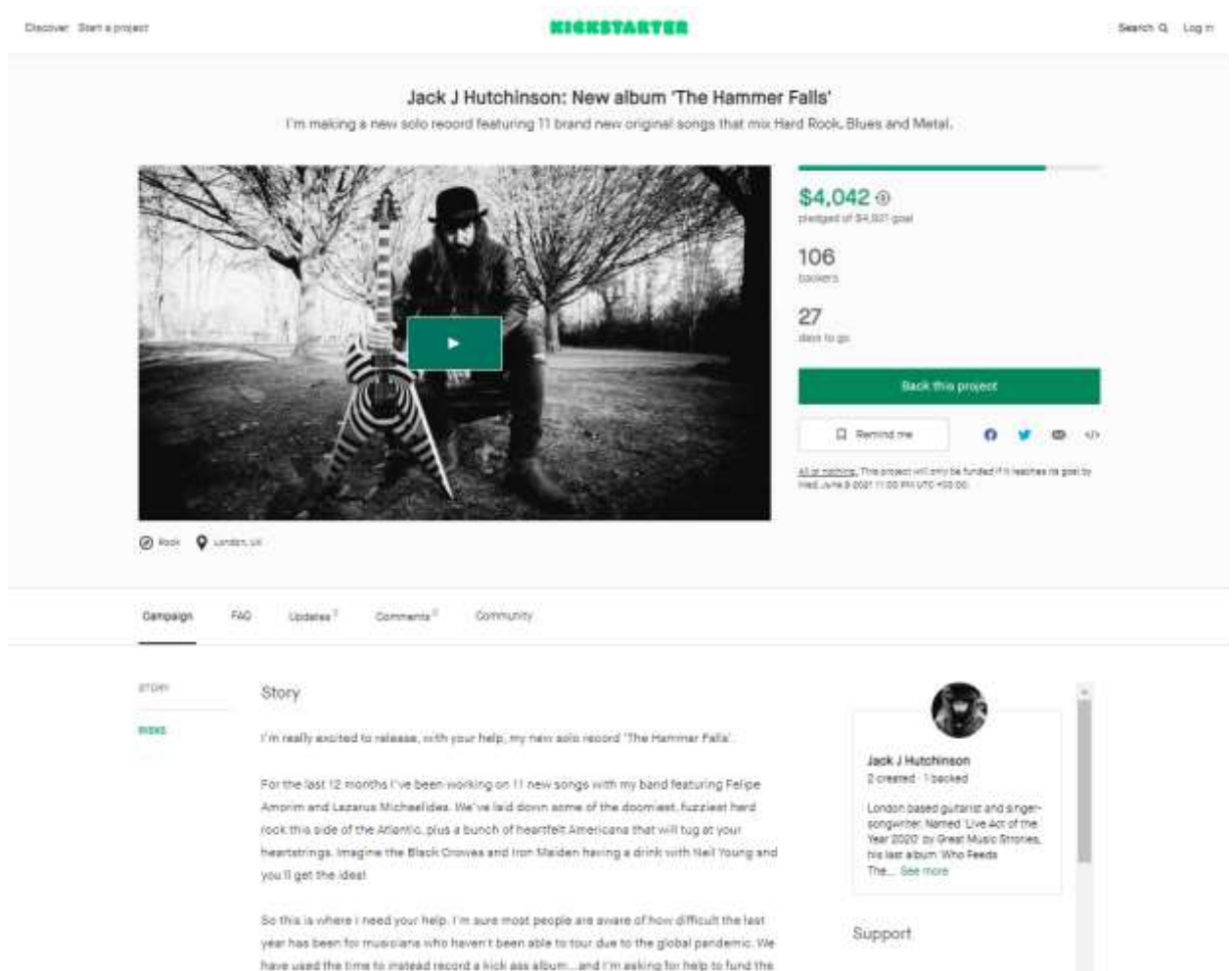


Рисунок 1.1 – Перегляд інформації про музичний проєкт на платформі
Kickstarter

Вебзастосунок надає наступні функціональні можливості:

- перегляду найпопулярніших проєктів;
- пошуку проєктів за їх назвами або категоріями;
- перегляду проєктів за категоріями, з яких винесено в основні мистецтво, ілюстрації, дизайн, фільми, музику, ігри, харчування, видавництво;
- перегляду проєктів за спеціальними переліками, що включають тренди, нещодавно створені, проєкти, які реалізуються поруч з користувачем, проєкти, що отримали фінансування;
- за кожною категорією проєктів надається перелік конкретних тегів, за якими можна також відобразити проєкти;
- створення нового проєкту, при цьому окрім ініціатора (автора) проєкту можна також визначати інших учасників проєкту;
- фінансування проєкту, при чому надається набір варіантів такого фінансування: перерахування заданої самим користувачем суми коштів, перерахування програмно зазначеної суми коштів, за яку передбачається отримання відповідних винагород (матеріалів, пов'язаних з проєктом, самих створених продуктів, в залежності від суми такого фінансування);
- створення коментарів за проєктами, які користувач профінансував;
- внесення проєкту в лист нагадування;
- надання інформації про поточні результати збору коштів на проєкт, включаючи заплановану суму, зібрану суму коштів на поточний момент, кількість внесків.

Кожен проєкт має детальний опис, окрім того за проєктом можуть виконуватися оновлення, що дозволяє відображати динамічні дані, а не тільки статичні протягом всього строку збору коштів.

Платформи такого типу мають певний недолік: платформі важко зібрати початкових ініціаторів, адже зібрати потрібну суму одразу може не вдатись за відсутності потрібної бази користувачів. Тобто подібна платформа повинна мати достатні фінансові ресурси початково, щоб за рахунок реклами

привернути достатньо велику базу потенційних спонсорів, тоді ідея з принципом «все або нічого» може виявитися дієвою.

Вебсервіс Patreon [6] (рис. 1.2) є ще одним відомим рішенням у цій галузі. Він не має таких засобів промотування як Kickstarter. Учасники часто рекламують себе на власних сторінках в соціальних мережах, привертаючи таким чином увагу. До того ж це загалом не інструмент фінансування проєктів, а інструмент постійного фінансування, коли найчастіше за рахунок підписки автори (музиканти) можуть збирати кошти для своєї діяльності.

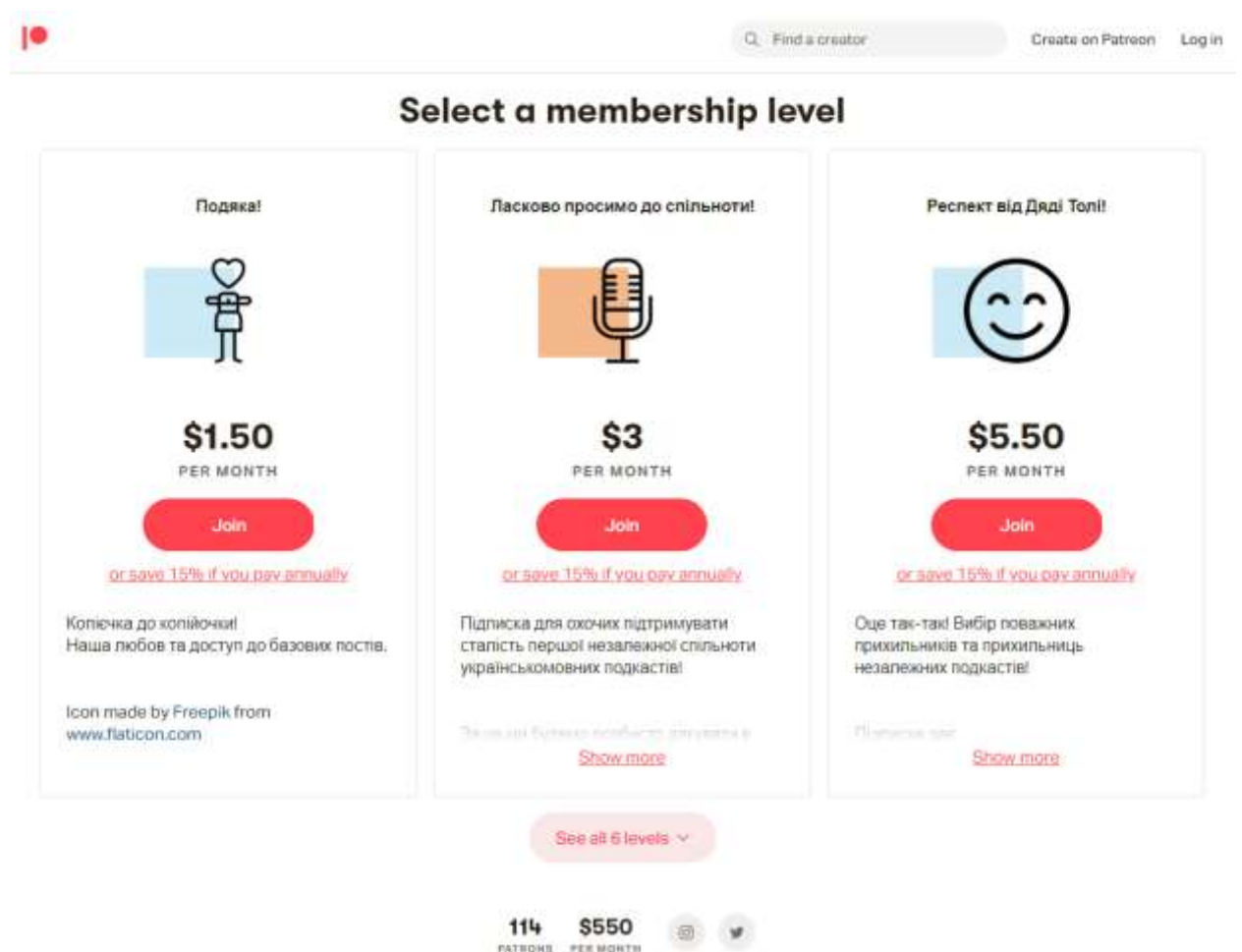


Рисунок 1.2 – Вибір тарифного плану для участі на платформі Patreon

Платформа дозволяє працювати не тільки музикантам, загалом, створюючи обліковий запис, користувача просять самостійно класифікувати свою роботу в одну з таких категорій:

- відео та фільми;
- малювання та живопис;
- підкасти;
- фотографія;
- наука;
- ремесла та саморобки;
- музика;
- комікси;
- анімація;
- ігри;
- комедія;
- освіта;
- танець і театр [7].

Щодо Patreon можна виділити наступне:

– пропонується три рівні ціноутворення, які приймають різні відсотки відрахувань: як Kickstarter, Indiegogo та більшість інших платформ для фінансування проєктів, базовий тариф Patreon Lite має типовий варіант збору коштів у 5 % від суми, тоді як Patreon Pro та Patreon Premium вимагають сплати 8 та 12 % відповідно, вищі рівні забезпечують додаткові переваги, такі як вдосконалені інструменти аналітики, у Patreon Premium надаються спеціальні майстер-класи, але для того щоб претендувати на участь у Patreon Premium кампанія повинна відповідати певним мінімумам фінансування;

– усі рівні аналогічно і Kickstarter використовують для обробки платежів сторонні платформи, які також несуть плату: плата за обробку платежів залежить від того, використовується PayPal або Stripe, але вони, як правило, складають відсоток пожертви і додатково фіксовану кількість центів, Patreon поєднує транзакції, коли це можливо, щоб зменшити витрати, існують також різні збори за міжнародні операції;

– після авторизації на інформаційній панелі користувача відображаються проєкти, які користувач підтримував, а також ті, за якими

користувач стежить, можна запросити іншого автора приєднатися до Patreon і отримати грошові винагороди за це, надається можливість відправлення повідомлень авторам проєктів;

– пропонує два варіанти отримання фінансування: можна попросити спонсорів або внести щомісячну плату, або плату за створення, у першому випадку спонсори встановлюють щомісячний бюджет (рекомендується використовувати опцію щомісяця, якщо планується випускати чотири або більше готових продуктів щомісяця), у другому випадку – випуск результатів відбувається у вигляді платних публікацій, які можуть бути у формі тексту, зображень, відео, аудіо чи їх комбінації, можна також встановити мінімальну плату для будь-якої з платних публікацій;

– після визначення способу оплати потрібно встановити цілі – вони є основними етапами збору коштів, які дозволяють додавати нові функції проєкту, такі як спеціальні ефекти для відео або видалення оголошень із програмного забезпечення чи програми, можна встановити декілька цілей і змінити їх у будь-який час, далі визначається винагорода для спонсорів залежно від того, скільки вони пожертвують, як і у випадку з Kickstarter та Indiegogo, Patreon дозволяє обмежити всі або деякі винагороди певною кількістю спонсорів;

– після того, як сторінка опублікована, можна додавати публікації на свою сторінку автора: кожную публікацію можна встановити як повністю загальнодоступною, так і лише для спонсорів або для спонсорів на різних рівнях винагороди, на інформаційній панелі можна відстежувати внески, можна переглянути, скільки меценатів є на кожен місяць, можна переглянути коментарі спонсорів, є також звіти про виплати, відмови (скасовані застави), повернення коштів [7]-[9].

Дана платформа зручна для організації постійного фінансування, але для збору коштів на проєкт вона є не зовсім придатною, оскільки швидше придатна для презентації і стягування плати за готові продукти, а не для збору коштів на створення продукту.

Indiegogo – ще одна з оригінальних платформ для громадського фінансування зокрема музичних проєктів, загалом послуги більше пристосовані для таких митців як музиканти, кінорежисери, вона схожа на Kickstarter, також дозволяє розпочати роботу безкоштовно, при створенні проєктів створюється відповідна сторінка та відео, що пояснюють проєкт, далі потрібно встановити мету та кінцевий термін кампанії, існує опція для проєктів, за якої є можливість збирати свої прибутки, навіть якщо не було досягнуто мету (Indiegogo використовує збір в 9 %, якщо цього не було зроблено, і 4%, якщо це було зроблено), загалом рішення оптимізоване для музикантів [1].

Основною відмінністю програмного забезпечення, що розробляється, є ідея поетапного збору коштів, до того ж це програма, яка буде направлена на роботу саме з музичними проєктами.

У табл. 1.1 виконано порівняння даних програмних платформ з програмним забезпеченням, яке планується розробити в роботі.

Таблиця 1.1 – Порівняння програмних аналогів

Параметр	Kickstarter	Patreon	Рішення, що розробляється
Основна одиниця	Проект	Автор	Проект
Основна концепція збору і використання коштів	Все або нічого	Підписка	Етапи
Підтримка музичних проєктів	Є	Є	Є
Платежі за перерахування коштів	Так	Так	Ні

1.3 Вимоги до програмного забезпечення

При реалізації програмного забезпечення потрібно враховувати наступні вимоги:

- забезпечення доступу до програми користувачами з різних пристроїв;
- автоматизація збору коштів на випуск альбомів, організацію концертів;
- виконання музичних проєктів реалізується поетапно з відповідним поетапним фінансуванням;
- кожен музичний проєкт має куратора;
- забезпечення платформи для взаємодії музикантів з потенційними спонсорами їх проєктів.

Основною особливістю програмного забезпечення, розробка якого виконується, порівняно з програмними аналогами є забезпечення захищеного фінансування. Захист організується зі сторони самої платформи. Під захистом розуміється закриття повної суми фінансування від музикантів. Для цього спонсор перелічує всю суму, а вона відправляється музиканту поетапно. Перехід від одного етапу до іншого відбувається на основі роботи куратора музичного проєкту. Тільки якщо куратор підтвердив виконання попереднього етапу, відбувається перехід до наступного, і музиканти отримують наступну порцію коштів.

Дана ідея підтримує бажання спонсорів долучитися до реалізації проєктів, але з іншого боку захищає їх від недобросовісних музикантів, які потенційно можуть отримати кошти і не створити потрібний продукт. Замість того, щоб в такому випадку відкочуватися до початкового стану, що пов'язано з проблемами взаємодії з музикантами, забороняється перехід до наступного етапу. За рахунок такої захищеності інтересів спонсорів передбачається, що програма зможе привернути додаткову увагу на тлі існуючих програмних аналогів.

1.4 Висновки за розділом 1

У першому розділі розглянуто проблему поетапного громадського фінансування музичних проєктів: розглянуто особливості проблеми як загалом, так і на основі існуючих програмних рішень, серед яких основними є Kickstarter та Patreon, проаналізовано їх функціональні можливості, виділено недоліки, виконано порівняння цих рішень з рішенням, яке розробляється в роботі, виділено вимоги до програми.

Завдання дипломної кваліфікаційної роботи:

- розробка технічного завдання і визначення вимог до програми;
- розробка архітектури програми;
- розробка і тестування програми.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Вибір технології розробки

Розробку програмного забезпечення потрібно виконувати у вигляді вебзастосунка, до якого зможе через мережу Інтернет отримувати доступ будь-який користувач, що дозволить впровадити програмне забезпечення одразу після створення без необхідності створювати застосунки в подальшому для додавання аудиторії користувачів.

Популярною технологією веброзробки на даний момент є розробка на основі рушія Node.JS. Альтернативою даному рішенню розглядається вебфреймворк Django [10]-[11].

Результати порівняння представлено в табл. 2.1.

Таблиця 2.1 – Порівняння технологій розробки

Параметр	Django	Node.JS
Мова програмування	Python	JavaScript
Проблеми з безпекою	Ні	Можливі
Вбудовані шаблони	Так	Ні
Швидкість розробки	Так	Ні

У результаті порівняння було обрано вебфреймворк Django для проєктування і реалізації програми, що ґрунтується на наступному:

- мова програмування Python [12] є простішою у засвоєнні і дозволяє створювати програмний код, який легше створювати і підтримувати;
- за рахунок мови Python швидкість розробки вища саме у Django-проєктах, до того ж час, який проходить з початку розробки до запуску першої версії застосунку за однакових умов менший саме у Django-проєктах;

– обидва рішення використовуються на практиці, підтримують роботу відомих рішень, проте засоби організації безпеки у них дещо різняться: за наявності великого досвіду Node.JS також дозволяє створювати вебзастосунки без дірок у безпеці, та при цьому потрібно спочатку отримати цей досвід, інакше рішення, які створюються, можуть мати проблеми з безпекою, у той же час рішення Django одразу інтегрують якісні засоби щодо забезпечення безпеки у свою структуру, вимагається менше знань для того, щоб не заподіяти шкоди даним, з якими працює програма;

– Node.JS має більшу свободу щодо організації формування макету сторінок, в той же час Django має готовий засіб у вигляді двигуна шаблонів, що зменшує свободу використання існуючих засобів, проте є надійним рішенням, яке не потребує витрат часу на вибір потрібного рішення з набору існуючих [10]-[13].

2.2 Архітектура програмного забезпечення

Для побудови архітектури програмного забезпечення було використано архітектурний шаблон Model-View-Template (MVT), відносно якого можна виділити наступне:

– складається з трьох компонентів, які іменуються як Model (модель), View (уявлення) та Template (шаблон);

– три шари відповідають за різні складові проблеми і використовуються самостійно;

– на відміну від шаблону Model-View-Controller, Django самостійно організовує частину контролера (програмний код, який контролює взаємодію між моделлю та уявленням), залишаючи необхідним тільки визначення шаблону;

– під шаблоном розуміється файл HTML, змішаний з мовою шаблонів Django Template Language (DTL);

– модель допомагає взаємодіяти з базою даних: це рівень доступу до даних, який містить обов'язкові поля та поведінку даних, які зберігаються, модель – це клас Python, і вона не знає нічого про інші шари Django, моделі допомагають розробникам створювати, читати, оновлювати та видаляти об'єкти (операції CRUD) у вихідній базі даних, також вони містять бізнес-логіку, власні методи, властивості та інші речі, пов'язані з маніпулюванням даними;

– уявлення використовується для виконання бізнес-логіки та взаємодії з моделлю для перенесення даних та візуалізації шаблону: уявлення отримує дані з моделі, потім воно або надає кожному шаблону доступ до певних даних, що відображаються, або обробляє дані заздалегідь, уявлення приймає HTTP-запити, застосовує логіку, яку надають класи та методи Python, та забезпечує відповіді HTTP на запити клієнта;

– шаблон – це презентаційний шар, який повністю обробляє частину інтерфейсу користувача, це файли з HTML-кодом, який використовується для візуалізації даних, вміст цих файлів може бути статичним або динамічним, шаблон використовується лише для подання даних, оскільки в ньому немає бізнес-логіки;

– в Django уявлення описує, які дані представлені, при цьому уявлення зазвичай делегує частину відповідальності шаблону, який описує спосіб подання даних;

– розробник визначає модель, уявлення та шаблон, потім відображає їх у URL-адресу, а Django передає вебсторінку користувачеві;

– загалом вебзастосунок має дані, макет та логіку: модель працює з даними, уявлення працює з логікою, а шаблон працює з макетом [14].

Шаблон MVT закладений в основу Django-проектів, його компоненти та логіка взаємодії між ними представлена на рис. 2.1. Центральним компонентом організації взаємодії є уявлення, яке взаємодіє і з моделлю, і з шаблоном.

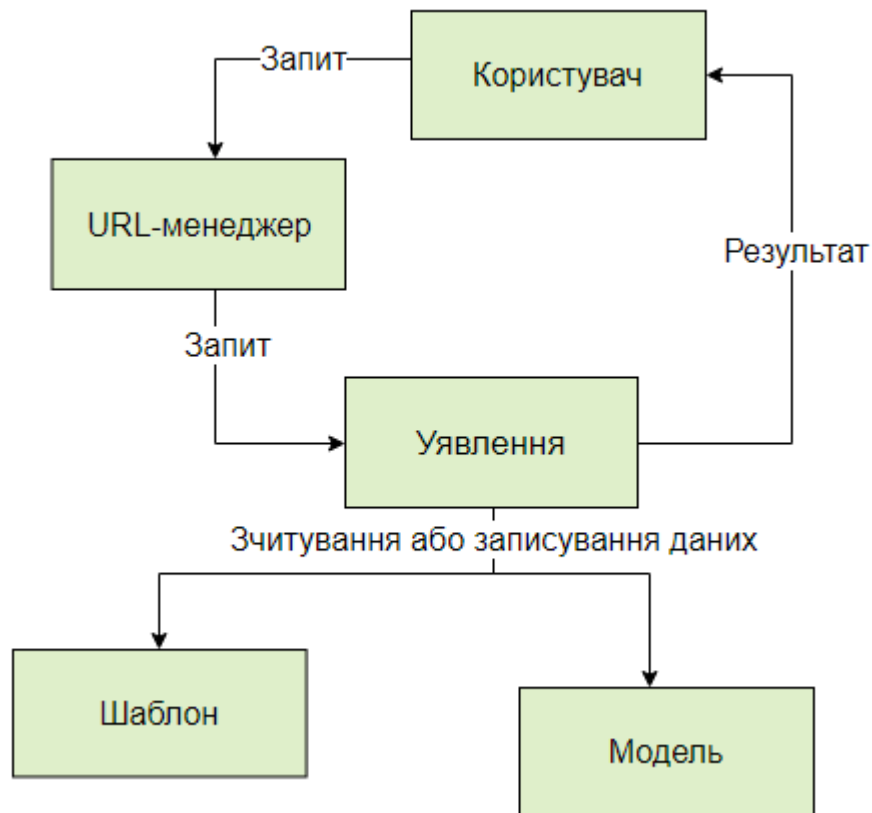


Рисунок 2.1 – Архітектурний шаблон MVT

2.3 Функціональні вимоги до програмного забезпечення

Для забезпечення концепції, сформульованої в першому розділі, програма повинна надавати наступні функціональні можливості:

- створення заявки на фінансування музичного проєкту;
- визначення етапів виконання і фінансування музичного проєкту;
- редагування створеної заявки на фінансування музичного проєкту;
- перегляд власних створених заявок на фінансування музичного проєкту;
- завершення збору коштів на музичний проєкт до початку першого етапу;
- підтвердження виконання етапу виконання музичного проєкту;
- підтвердження невиконання етапу виконання музичного проєкту;
- перерахування коштів на виконання музичного проєкту;
- відображення інформації про музичний проєкт;

- внесення коментарів на профінансований користувачем музичний проєкт;
- пошук музичних проєктів;
- відображення власних фінансувань музичних проєктів;
- перегляд сум, внесених на фінансування музичного проєкту;
- реєстрація облікових записів користувачів;
- визначення для користувача ролі заявника або куратора;
- авторизація облікового запису;
- відображення музичних проєктів, які отримали найбільше фінансування;
- відображення музичних проєктів за часом створення;
- відображення профінансованих музичних проєктів;
- відображення музичних проєктів, які знаходяться на фінансуванні і отримали його перший етап;
- відображення музичних проєктів, які не отримали фінансування;
- відображення музичних проєктів, які отримали початкове фінансування, але не були завершені.

Для забезпечення даних функціональних можливостей усіх користувачів було розподілено за наступними ролями:

- адміністратор (рис. 2.2) – особа, яка здійснює керування ролями інших користувачів;
- заявник (рис. 2.3) – особа, яка формує заявки на фінансування музичних проєктів і отримує фінансування;
- куратор – особа, яка реалізує підтвердження виконання або невиконання етапів проєкту;
- авторизований користувач – особа, яка має можливість фінансувати і коментувати заявки на музичні проєкти;
- неавторизований користувач – особа, яка може переглядати тільки загальну інформацію про музичні проєкти.



Рисунок 2.2 – Діаграма прецедентів адміністратора

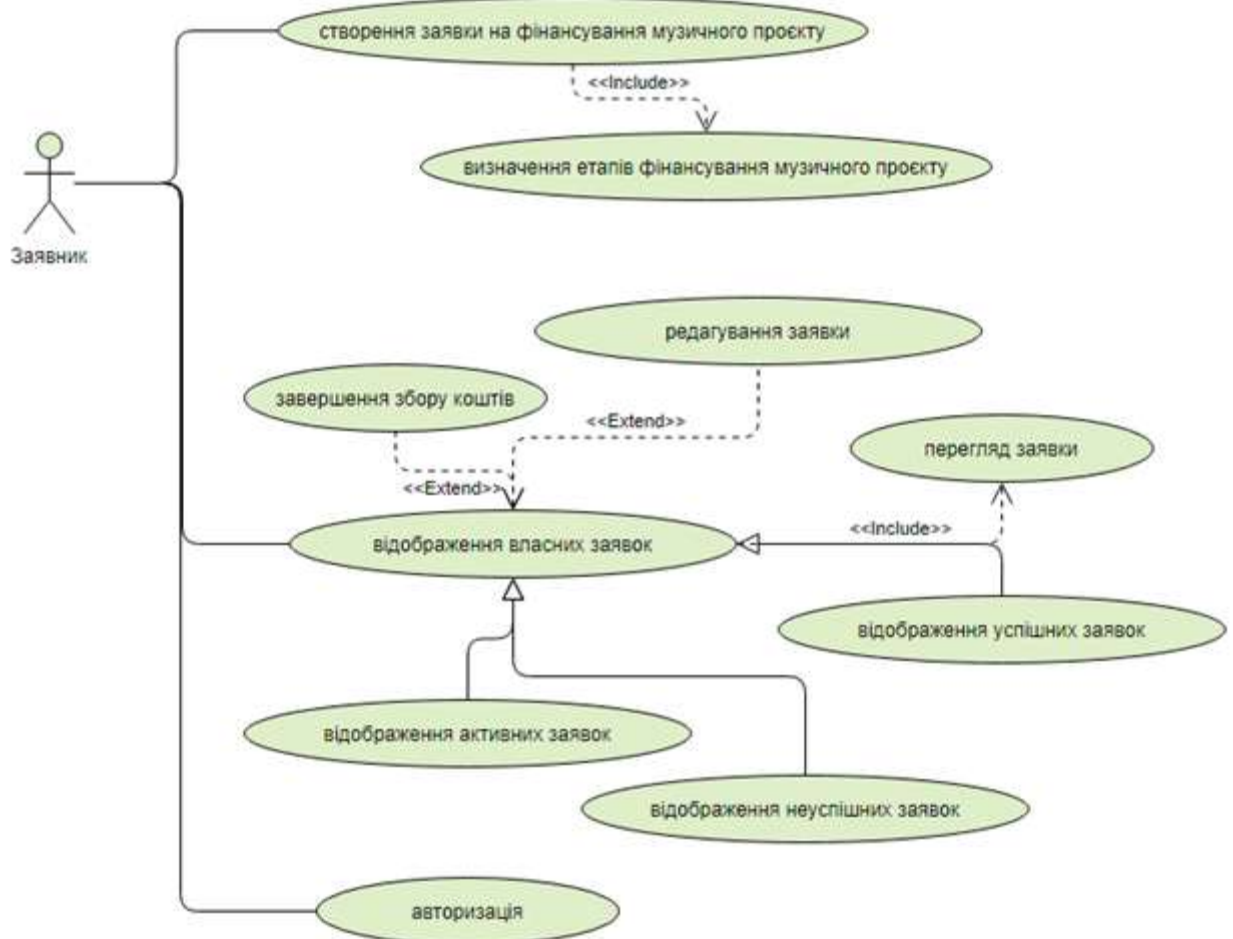


Рисунок 2.3 – Діаграма прецедентів заявника

Для куратора (рис. 2.4) та неавторизованого користувача (рис. 2.5) було створено діаграми прецедентів.

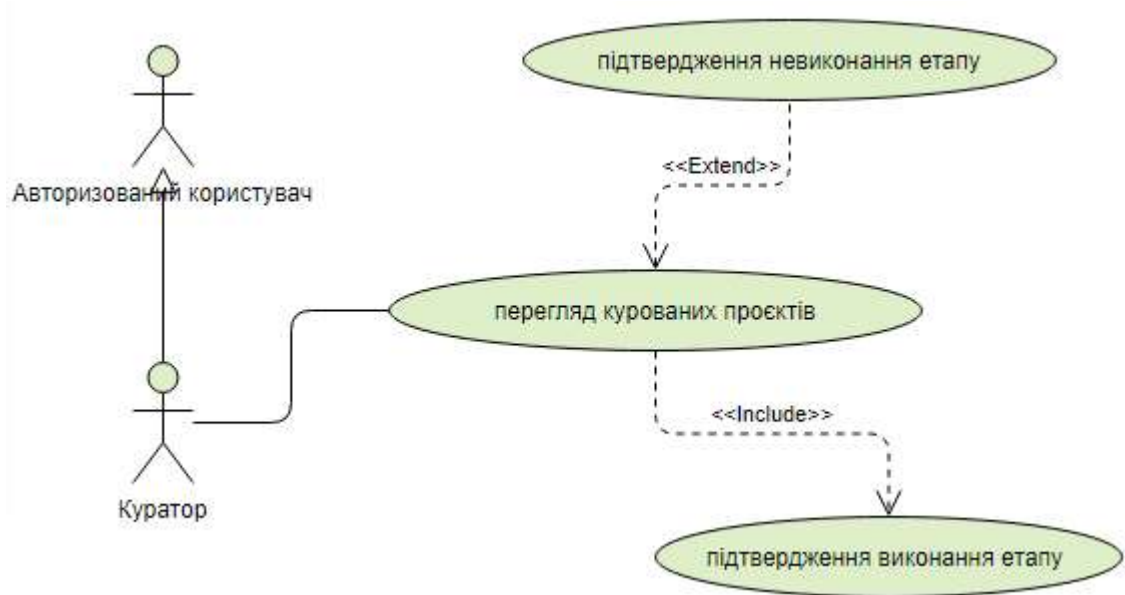


Рисунок 2.4 – Діаграма прецедентів куратора

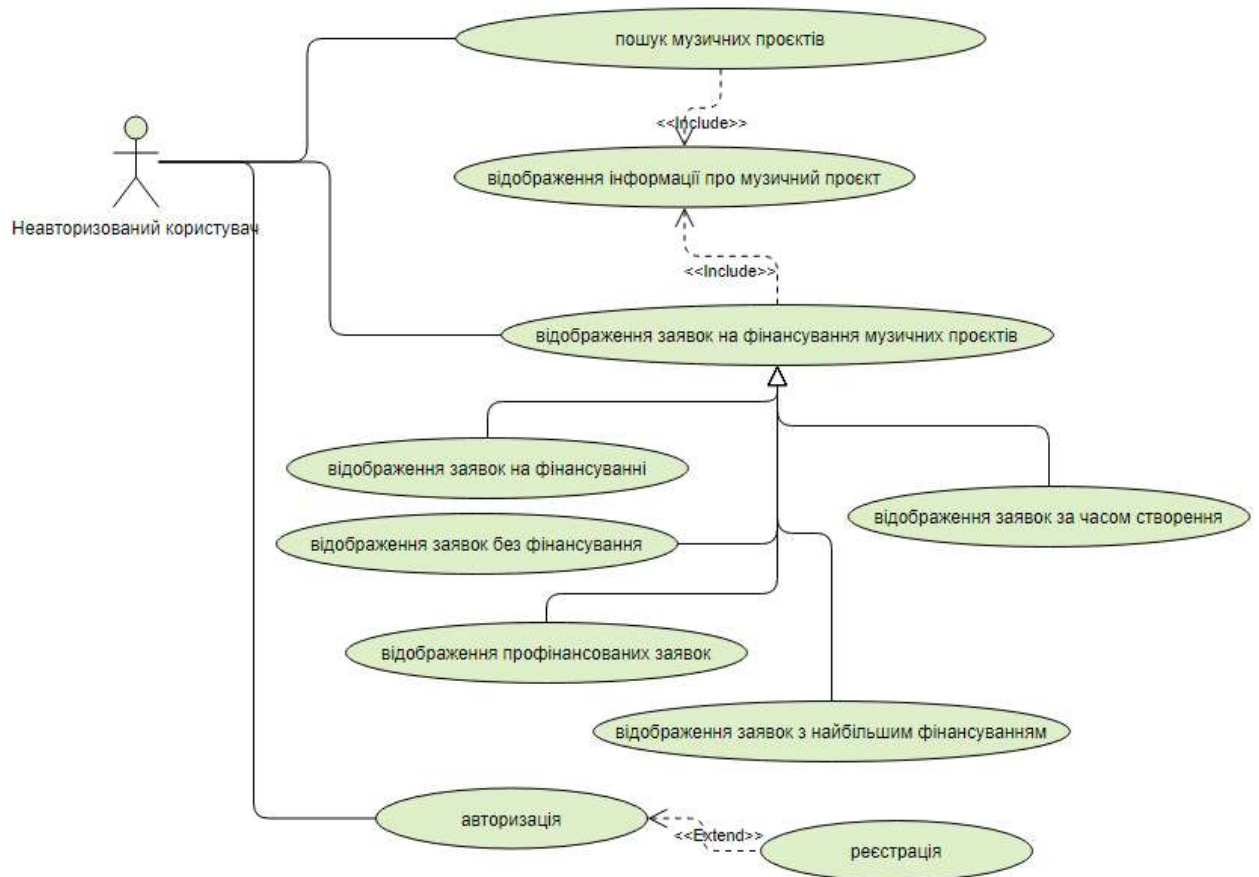


Рисунок 2.5 – Діаграма прецедентів неавторизованого користувача

На рис. 2.6 представлено діаграму прецедентів авторизованого користувача.



Рисунок 2.6 – Діаграма прецедентів авторизованого користувача

2.4 Проєктування структури даних

Системою керування базами даних для організації роботи з даними було визначено MySQL.

Виділені сутності предметної області:

- заявка на фінансування музичного проєкту MusProject;
- етап виконання музичного проєкту MusProjectStage;
- платіж Payoff;
- тег Tag;
- обліковий запис користувача User.

Таблиця MusProject має наступні поля:

- id – номер заявки на фінансування музичного проєкту;
- title – назва музичного проєкту;
- description – опис музичного проєкту;
- created – дата створення заявки на фінансування музичного проєкту;
- costs – загальна сума, необхідна для фінансування музичного проєкту;
- gathcosts – зібрана сума на фінансування музичного проєкту;
- status – стан збору коштів;
- video – відео для ілюстрації музичного проєкту;

- `deadline` – дата, до якої має бути завершено збір коштів на фінансування музичного проєкту;
- `authorId` – номер облікового запису користувача, який є автором музичного проєкту;
- `curatorId` – номер облікового запису користувача, який є куратором музичного проєкту.

Поля `authorId` і `curatorId` є зовнішніми ключами з таблиці `User` і визначають користувачів.

Таблиці `MusProject` і `User` зв'язані зв'язком один-до-багатьох, оскільки кожен користувач може бути куратором або автором багатьох музичних проєктів, але кожен музичний проєкт може мати тільки одного автора і тільки одного куратора.

Таблиця `MusProjectStage` має наступні поля:

- `id` – номер етапу виконання музичного проєкту;
- `musprojectId` – номер музичного проєкту, до якого належить етап;
- `deadline` – дата, до якої має бути завершено виконання етапу;
- `costs` – сума, потрібна для виконання даного етапу музичного проєкту;
- `status` – стан музичного проєкту;
- `title` – назва етапу музичного проєкту;
- `description` – опис етапу музичного проєкту;
- `order` – порядковий номер виконання етапу серед всіх етапів музичного проєкту.

Поле `musprojectId` є зовнішнім ключем з таблиці `MusProject`.

Таблиці `MusProject` і `MusProjectStage` зв'язані зв'язком один-до-багатьох, оскільки кожен музичний проєкт може мати багато етапів, а кожен етап здійснюється в межах тільки одного проєкту.

Таблиця `Payoff` має наступні поля:

- `id` – номер платежу;
- `created` – дата створення платежу;

- banktransaction – номер банківської транзакції;
- account – банківський рахунок, з якого виконано сплату;
- status – стан платежу;
- costs – сума платежу;
- userId – номер облікового запису користувача, який виконав платіж;
- musprojectId – номер музичного проєкту, на який направлено платіж.

Поле musprojectId є зовнішнім ключем з таблиці MusProject.

Поле userId є зовнішнім ключем з таблиці User.

Таблиці MusProjectStage і Payoff зв'язані зв'язком один-до-багатьох, оскільки для кожного етапу може бути виконано багато платежів, але кожен платіж направлений тільки на один етап.

Таблиці User і Payoff зв'язані зв'язком один-до-багатьох, оскільки кожен користувач може виконати багато платежів на фінансування різних заявок, але кожен платіж може мати тільки одного автора.

Таблиця Tag має наступні поля:

- id – номер тегу;
- name – тег.

Таблиця MusProjectTag реалізує зв'язок багато-до-багатьох між таблицями MusProject і Tag, оскільки кожен тег може характеризувати різні музичні проєкти, але кожен музичний проєкт може характеризуватися багатьма тегами. Для реалізації цієї таблиці було введено зовнішніми ключами номери відповідних музичних проєктів та тегів.

Таблиця має наступні поля:

- id – номер;
- musprojectId – заявка на фінансування музичного проєкту;
- tagId – тег.

Поле musprojectId є зовнішнім ключем з таблиці MusProject.

Поле tagId є зовнішнім ключем з таблиці Tag.

Відповідну схему бази даних приведено на рис. 2.7.

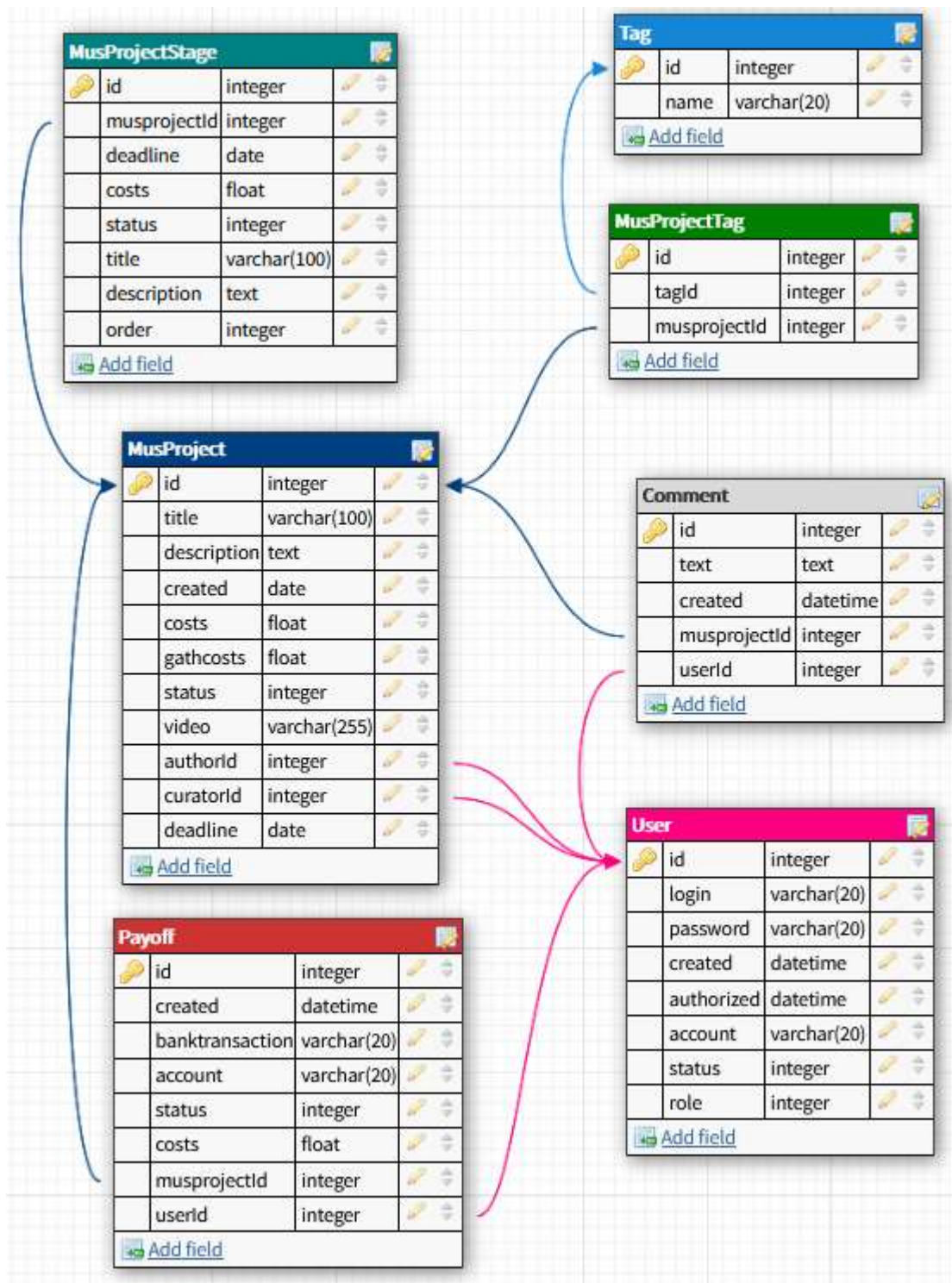


Рисунок 2.7 – Схема бази даних

Таблиця Comment має наступні поля:

- id – номер коментаря;
- text – текст коментаря;
- created – дата створення облікового запису;

- `userId` – номер облікового запису користувача, який створив коментар;
- `musprojectId` – номер музичного проєкту, до якого адресовано коментар.

Поле `musprojectId` є зовнішнім ключем з таблиці `MusProject`.

Поле `userId` є зовнішнім ключем з таблиці `User`.

Таблиці `User` і `Comment` зв'язані зв'язком один-до-багатьох, оскільки кожен користувач може створити коментарі до багатьох музичних проєктів, але кожен коментар створений тільки одним користувачем.

Таблиці `MusProject` і `Comment` зв'язані зв'язком один-до-багатьох, оскільки кожен музичний проєкт може мати багато коментарів, але кожен коментар адресований тільки одному музичному проєкту.

Таблиця `User` має наступні поля:

- `id` – номер облікового запису;
- `login` – логін облікового запису;
- `password` – пароль облікового запису;
- `created` – дата створення облікового запису;
- `authorized` – дата останньої авторизації облікового запису;
- `account` – банківський рахунок користувача;
- `status` – стан облікового запису;
- `role` – роль користувача.

2.5 Висновки за розділом 2

Результати виконання даного розділу дипломної кваліфікаційної роботи полягають в наступному:

- виконано вибір мови програмування Python і вебфреймворку Django для реалізації програми;
- визначено архітектуру програмного забезпечення на основі архітектурного шаблону MVT;

- визначено функціональні вимоги до програми;
- побудовано діаграми прецедентів для роботи адміністратора, заявника, куратора, авторизованого та неавторизованого користувачів з програмою;
- визначено структуру даних за допомогою схеми бази даних.

3 РОЗРОБКА ПРОГРАМИ

3.1 Опис послідовності роботи програми

Оскільки з програмою взаємодіють користувачі в різних ролях, а кожна роль вимагає відповідного набору сценаріїв взаємодії, то для узагальнення послідовності роботи програми з різними користувачами було вирішено розглянути цю послідовність за допомогою стану основного об'єкту предметної області – музичного проєкту. Під час збору коштів його можна розглядати як заявку на фінансування музичного проєкту, а після його завершення і переходу до виконання – як музичний проєкт.

Музичний проєкт може знаходитися в одному з наступних станів (рис. 3.1):

- на фінансуванні: у даному стані музичний проєкт знаходиться тоді, коли створено заявку на фінансування і користувачі мають можливість надіслати кошти на його виконання;
- профінансований: у даному стані музичний проєкт знаходиться тоді, коли збір коштів було завершено успішно, при цьому виконання проєкту ще не розпочалось;
- непрофінансований: у даному стані музичний проєкт знаходиться тоді, коли збір коштів було завершено, але виявилось, що зібрана сума коштів недостатня для його виконання, також до цього стану проєкт може потрапити, якщо під час збору коштів автор зупинив цей процес;
- виконуваний: у даному стані музичний проєкт знаходиться тоді, коли розпочалося виконання музичного проєкту, з проєктом починає взаємодіяти куратор;
- неуспішний: у даному стані музичний проєкт знаходиться тоді, коли після збору потрібної суми коштів виконати проєкт не вдалося, про що наявна відповідна відмітка куратора проєкту;

– успішний: у даному стані музичний проєкт знаходиться тоді, коли успішно завершився не тільки збір коштів, але і виконання всіх етапів проєкту, кожний з яких підтвердив куратор.

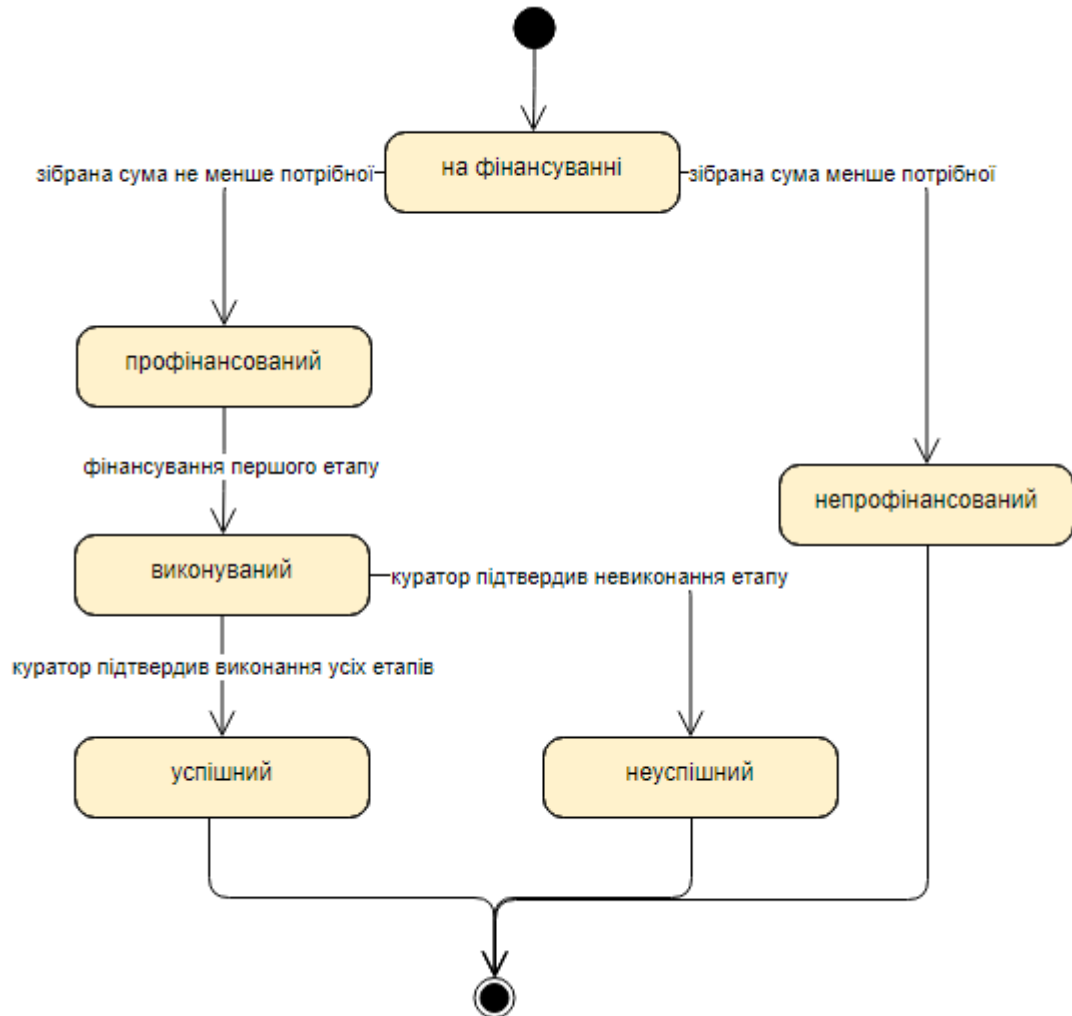


Рисунок 3.1 – Діаграма станів музичного проєкту

3.2 Опис реалізації програми

Під час реалізації програмного забезпечення використано об'єктно-реляційне відображення Django, відповідно створення бази даних виконувалось на основі коду, тобто класів моделей, які визначали структуру таблиць.

У процесі реалізації програмного забезпечення було створено наступні класи моделей.

Клас MusProject визначає музичний проєкт (початково заявку на музичний проєкт), який містить наступні поля:

- title – назва музичного проєкту, що представлена об'єктом класу CharField, в параметрах конструктора якого визначена максимальна довжина символьного поля 100 символів;

- description – опис музичного проєкту, що представлений текстовим полем, яке є об'єктом класу TextField;

- created – дата створення заявки на фінансування музичного проєкту, що представлено полем дати, яке є об'єктом класу DateField;

- deadline – дата, до якої має бути завершено збір коштів на проєкт (у випадку якщо не буде завершено збір коштів, то всі платежі мають повернутися на рахунки користувачів, які їх виконали), яке є об'єктом класу DateField;

- costs – сума, потрібна на фінансування заявки музичного проєкту, що представлено дійсним (числом з плаваючою крапкою) полем, яке є об'єктом класу FloatField;

- gathcosts – зібрана сума на фінансування заявки музичного проєкту, що представлено дійсним (числом з плаваючою крапкою) полем, яке є об'єктом класу FloatField;

- status – стан музичного проєкту, що представлений цілочисельним полем невеликого діапазону, яке є об'єктом класу SmallIntegerField;

- video – шлях до відеофайлу, що представлений об'єктом класу CharField з максимальною довжиною символьного поля 255 символів;

- author – автор музичного проєкту, що представлений зовнішнім ключем таблиці бази даних, для визначення поля створено зв'язок між таблицями за допомогою класу ForeignKey з визначенням класу User для зв'язку;

– curator – куратор музичного проєкту, що представлений зовнішнім ключем таблиці бази даних, для визначення поля створено зв'язок між таблицями за допомогою класу ForeignKey з визначенням класу User для зв'язку;

– tags = models.ManyToManyField(Tag) – поле, яке визначає зв'язок багато-до-багатьох і містить перелік тегів, якими характеризується музичний проєкт.

Клас містить властивість, створену визначенням методу done, що виконує обчислення відсотку зібраних коштів стосовно коштів, які потрібно зібрати:

```
@property
def done(self):
    res = self.gathcosts / self.costs
    return res
```

Окрім того було створено метод Meta, в якому визначено порядок стандартного сортування музичних проєктів. У складі класу було реалізовано наступні методи.

Метод __str__ визначає спосіб оброблення об'єктів класу як рядка, повертаючи при цьому назву музичного проєкту.

Метод get_absolute_url повертає посилання на детальне представлення конкретного музичного проєкту, номер якого міститься в параметрах методу через self.id.

Метод endfinance призначений для завершення збору коштів на фінансування, змінюючи стан музичного проєкту на «профінансований», якщо зібрана сума більша або дорівнює потрібній, або на «непрофінансований», якщо зібрана сума менша за потрібну. Метод повертає значення True або False в залежності від того, чи вдалося виконати операцію, чи ні.

Метод `stopfinance` завершує дочасно збір коштів на фінансування проєкту, встановлюючи стан музичного проєкту «непрофінансований».

Метод `start` визначає початок виконання проєкту після завершення збору коштів на його фінансування і встановлює стан проєкту «виконуваний».

Метод `successend` визначає успішне завершення виконання проєкту і встановлює стан проєкту «успішний».

Метод `stop` визначає незавершення виконання одного з етапів музичного проєкту і встановлює стан проєкту «неуспішний».

Для наповнення шаблонів вебзастосунку даними музичних проєктів були створені наступні класи контролерів музичних проєктів.

Клас `MusProjectsView` створює перелік музичних проєктів, розділений на порції по 10 на сторінці, сортуючи цей перелік в напрямку, заданому параметром GET-запиту. Сортування виконується в методі `get_queryset`, який визначає відповідь на запит, яка повертається для формування вебсторінки на основі шаблону. Сортування відбувається за алфавітним порядком, в протилежному алфавітному порядку, за зменшенням дати створення, за збільшенням дати створення, за зменшенням відсотку зібраних коштів, за збільшення відсотку зібраних коштів, перед цим реалізується вибірка всіх музичних проєктів:

```
self.ordering = self.request.GET.get('ordering')
data = MusProject.objects.all()
if self.ordering == 0:
    data = data.order_by('title')
elif self.ordering == 1:
    data = data.order_by('-created')
elif self.ordering == 2:
    data = data.order_by('created')
elif self.ordering == 3:
```

```

        data = data.order_by('title')
    elif self.ordering == 4:
        data = data.order_by('-done')
    elif self.ordering == 5:
        data = data.order_by('done')

```

Клас `OwnMusProjectsView` створює перелік власних музичних проєктів авторизованого користувача (заявника), організований шляхом наслідування класів `LoginRequiredMixin`, оскільки для виконання такого відображення потрібно бути авторизованим в системі, та `ListView`, оскільки дані, що відображаються, будуть представляти собою перелік.

Клас `OwnMusProjectView` надає автору музичного проєкту повну інформацію про музичний проєкт з номером, який задається в адресі сторінки проєкту. Окрім інформації з відповідного об'єкту в контекст сторінки передається перелік етапів музичного проєкту, перелік платежів за проєктом та перелік коментарів до проєкту.

Клас `MusProjectView` також надає деталізовану інформацію про музичний проєкт, але при цьому даний клас використовується не тільки авторами проєктів, тому замість повного набору платежів надається перелік користувачів, які внесли суми в фінансування музичного проєкту.

Клас `OwnSuccessMusProjectsView` надає перелік власних музичних проєктів, збір коштів на які завершився успішно.

Клас `OwnFailedMusProjectsView` надає перелік власних музичних проєктів, збір коштів яких не дав результатів, сортуючи проєкти за віддаленням дати створення від поточної.

Клас `OwnUndoneMusProjectsView` надає перелік власних музичних проєктів, які під час виконання зазнали невдачі.

Клас `OwnDoneMusProjectsView` надає перелік власних музичних проєктів, які завершилися успішно як на стадії збору коштів, так і за результатами виконання музичного проєкту.

Клас `OwnInprocessProjectsView` надає перелік власних музичних проєктів, які виконуються в даний момент.

Клас `OwnFinanceProjectsView` надає перелік власних музичних проєктів, які в даний момент знаходяться на стадії збору коштів, сортуючи його за зменшенням відсотку збору коштів.

Клас `CuratorMusProjectsView` надає перелік музичних проєктів, для яких куратором є авторизований в даний момент куратор.

Клас `SearchMusProjectsView` забезпечує пошук музичних проєктів за назвою або її частиною (додатково за потреби виконуючи пошук в опису), за інтервалом дати створення музичного проєкту, за потрібною для збору сумою коштів та за зібраною на даний момент сумою коштів, сортуючи створений перелік за заданим напрямком. Усі параметри передаються через GET-запит.

Клас `MusProjectUpdateView` виконує підтримку редагування музичного проєкту за допомогою відповідної форми.

Клас `TagMusProjectsView` надає перелік музичних проєктів, що відповідають заданому тегу.

Клас `MaxFinMusProjectsView` надає перелік 100 найбільших за зібраними сумами музичних проєктів.

Клас `SuccessMusProjectsView` надає перелік музичних проєктів, сума зібраних коштів на які перевищила потрібну.

Клас `FailedMusProjectsView` надає перелік музичних проєктів, на які було зібрано коштів менше ніж потрібно.

Клас `UndoneMusProjectsView` надає перелік усіх музичних проєктів, які не досягли завершення під час виконання.

Клас `DoneMusProjectsView` надає перелік усіх музичних проєктів, які успішно були виконані в результаті.

Клас `InprocessMusProjectsView` повертає перелік музичних проєктів, які в даний момент виконуються і ще не досягли успіху.

Клас `MusProjectCreateView` виконує підтримку створення музичного проєкту на основі наслідування класу `CreateView`, передаючи в контекст уявлення дані доступних тегів, визначаючи автора проєкту та встановлюючи дату створення проєкту.

Організацію платежів реалізовано за допомогою зовнішньої бібліотеки [15] на основі платіжної системи `LiqPay`.

Шаблони вебсторінок проєкту створені мовою DTL на основі вільного шаблону [16], який містить каскадні таблиці стилів, і включають зокрема наступні в папці для музичного проєкту `musproject`:

- `index.html` – шаблон відображення всіх заявок на фінансування;
- `own.html` – шаблон відображення всіх власних заявок;
- `detail.html` – шаблон відображення повної інформації музичного проєкту;
- `create.html` – шаблон вебсторінки створення нового музичного проєкту;
- `maxfinance.html` – шаблон перегляду переліку 100 перших заявок, що отримали найбільше фінансування;
- `undone.html` – шаблон перегляду проєктів, які не було виконано вчасно;
- `tag.html` – шаблон перегляду проєктів, які містять заданий тег;
- `owninproc.html` – шаблон перегляду власних заявок, які знаходяться у процесі виконання;
- `search.html` – шаблон пошуку заявок;
- `ownsucccess.html` – шаблон власних успішно завершених проєктів;
- `curator.html` – шаблон перегляду проєктів, якими керує куратор;
- `update.html` – шаблон редагування даних музичного проєкту.

Структуру створеного програмного забезпечення представлено на рис. 3.2, на нього винесено основні структурні елементи.

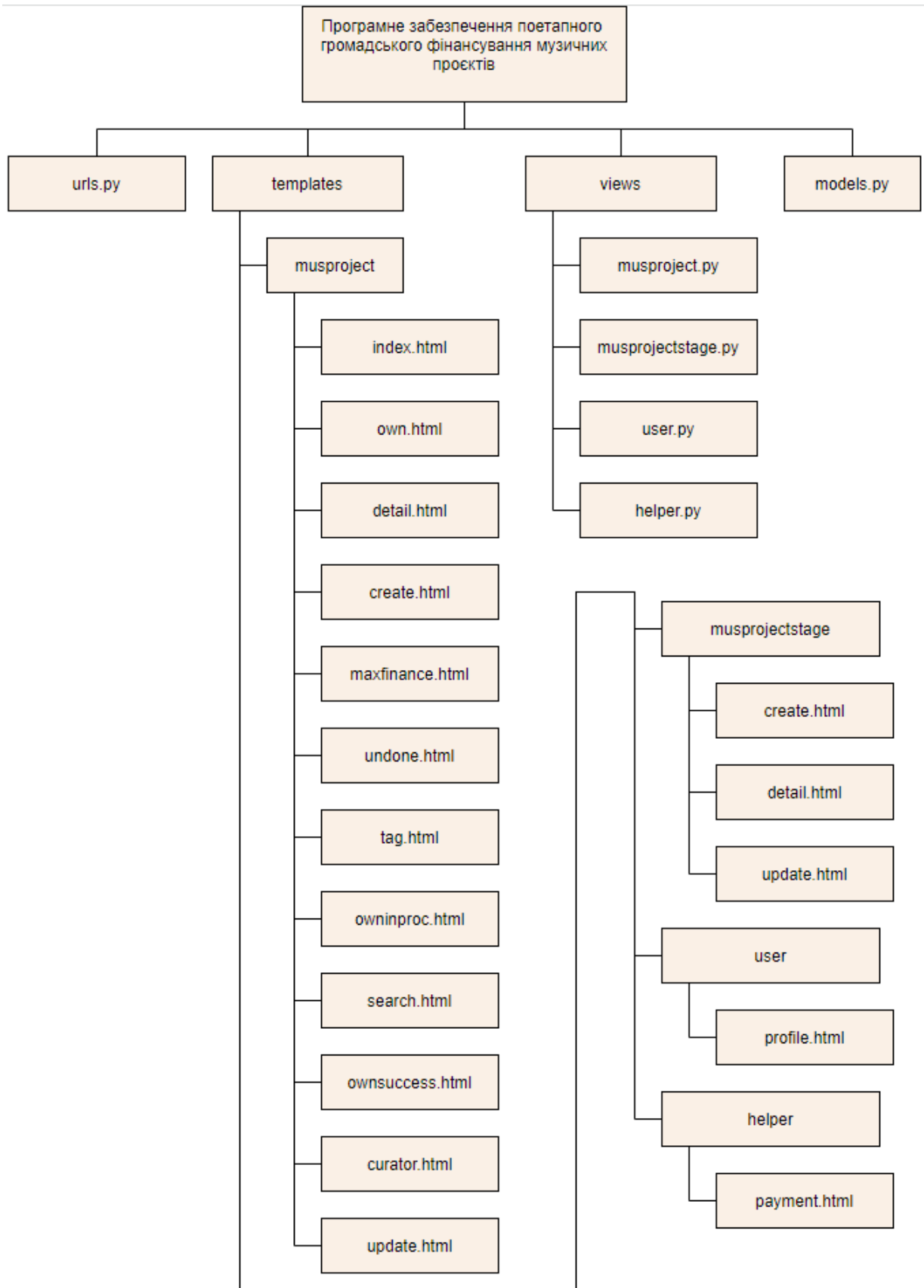


Рисунок 3.2 – Структура розробленої програми

Інші шаблони структуровані в наступні папки:

- musicstage – всі шаблони етапу музичного проєкту (створення create.html, редагування update.html, повної інформації detail.html);
- user – містить шаблони, пов'язані з користувачем: profile.html для керування профілем, payments.html для перегляду всіх платежів користувача;
- helper – усі інші шаблони, зокрема шаблон платежу payment.html.

3.3 Висновки за розділом 3

Результати виконання даного розділу дипломної кваліфікаційної роботи полягають в наступному:

- описано послідовність роботи програми на основі діаграми станів музичного проєкту;
- описано реалізацію програмного забезпечення з приведенням основних розроблених елементів;
- представлено структуру розробленого програмного забезпечення.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення програми

Програма призначена для організації підтримки громадського фінансування музичних проєктів, надаючи можливість створення заявок на фінансування музичних проєктів, керування цими заявками, створення власних внесків на фінансування музичних проєктів, які надаються проєкту поетапно, після того, як куратор проєкту підтвердив виконання попереднього етапу, і повертаються користувачу, якщо потрібну суму на фінансування не вдалося зібрати.

4.2 Умови виконання програми

Мінімальні вимоги до серверної сторони:

- жорсткий диск з вільним простором 1 Гб;
- оперативна пам'ять обсягом 4 Гб;
- процесор з 4 ядрами і частотою 2,8 ГГц.

Для виконання вимог до клієнтської сторони потрібний встановлений сучасний браузер з доступом до мережі Інтернет.

4.3 Робота з програмою

Робота з програмою розпочинається з введення в браузер адреси вебсервера, на якому розташовано вебзастосунок. Після цього користувачу відображається основна сторінка для неавторизованого користувача.

У випадку, якщо користувач бажає працювати в режимі адміністратор, він має вказати в якості адреси «/admin» після адреси вебсервера, тоді відкриється сторінка авторизації, де потрібно задати дані облікового запису адміністратора, після чого у разі успішної авторизації адміністратору відображається адміністративна панель, що є стандартною

адміністративною панеллю Django і дозволяє керувати ролями користувачів та керувати створеними музичними проєктами, зокрема визначити для проєкту його куратора, що є основною відповідальністю адміністратора.

Розглянемо основні засоби роботи звичайних користувачів.

Заявник має обмежене меню, яке дозволяє працювати тільки з засобами щодо власних заявок на фінансування музичних проєктів. Меню заявника, розташоване зліва на сторінці, включає наступні пункти:

- «Створити» – пункт меню, який дозволяє створювати нові заявки на фінансування музичних проєктів (рис. 4.1);

- розділ «Власні заявки», який включає підпункти «Усі заявки» для відображення усіх раніше створених власних заявок на фінансування, «На фінансуванні», що дозволяє переглянути тільки ті власні заявки, строк збору коштів на які не завершився, «Профінансовані», що дозволяє переглянути всі власні заявки, які отримали повне необхідне фінансування, «Непрофінансовані», що дозволяє переглянути всі власні заявки, які не набрали потрібну суму за відведений час, «Виконувані», що дозволяє переглянути перелік власних музичних проєктів, виконання яких розпочато, але не завершено, «Успішні», що дозволяє переглянути власні музичні проєкти, які були успішно виконано, «Неуспішні», що дозволяє переглянути власні музичні проєкти, які завершилися невдачею під час їх реалізації.

У заголовку сторінки розташовано рядок пошуку, який дозволяє виконати пошук серед власних заявок на фінансування за назвою. У правому куті заголовка знаходиться овал з логіном користувача, при натисненні на який відкривається сторінка профілю заявника. Для того щоб залишити власний обліковий запис, потрібно натиснути на посилання «вийти».

Заявка отримує фінансування тільки в тому випадку, якщо за відведений час було зібрано потрібну суму коштів. Якщо кошти не зібрано, то всі платежі повертаються назад користувачам, які їх здійснили.

ЯРМУЗФІН

Шукати

Вийти

Здійснюйте проекти з нами

Нова заявка на фінансування

Зберегти

ЗАГАЛЬНІ ДАНІ

Назва проєкту

Введіть назву

Опис проєкту

Введіть опис Вашого проєкту

Бюджет проєкту, грн.

Введіть суму для фінансування

Дедлайн

дд.мм.рррр

ЕТАПИ ПРОЕКТУ

НОМЕР	НАЗВА	СУМА	ДЕДЛАЙН
НЕМАЄ			

Назва етапу

Введіть назву

Сума

Дедлайн

Порядок

Сума фінансування

дд.мм.рррр

номер

Рисунок 4.1 – Створення заявки на фінансування музичного проєкту

Під час створення нової заявки на фінансування музичного проєкту заявник має задати для неї наступні групи параметрів:

- загальні дані, що включають назву проєкту, його опис, бюджет, який потрібен на фінансування проєкту, дедлайн збору коштів на фінансування проєкт;
- етапи проєкту, що включають таблицю з етапами проєкту, для

кожного з яких виводиться порядковий номер серед всіх етапів проєкту, його назва, сума, дедлайн виконання та кнопка редагування, якщо такий етап існує (у випадку відсутності будь-якого етапу відображається напис «немає»): для того щоб додати новий етап, потрібно в полях нижче задати назву етапу, суму, необхідну на його фінансування, дедлайн виконання етапу, порядковий номер та опис етапу, після чого натиснути на кнопку «Додати»;

– відео: файл відео, який буде використовуватися для промоції музичного проєкту, файл буде завантажений на сервер;

– теги: дозволяє додати необхідну кількість існуючих тегів до опису музичного проєкту.

Пункт меню «Усі заявки» відкриває сторінку перегляду всіх власних заявок на фінансування заявника (рис. 4.2).

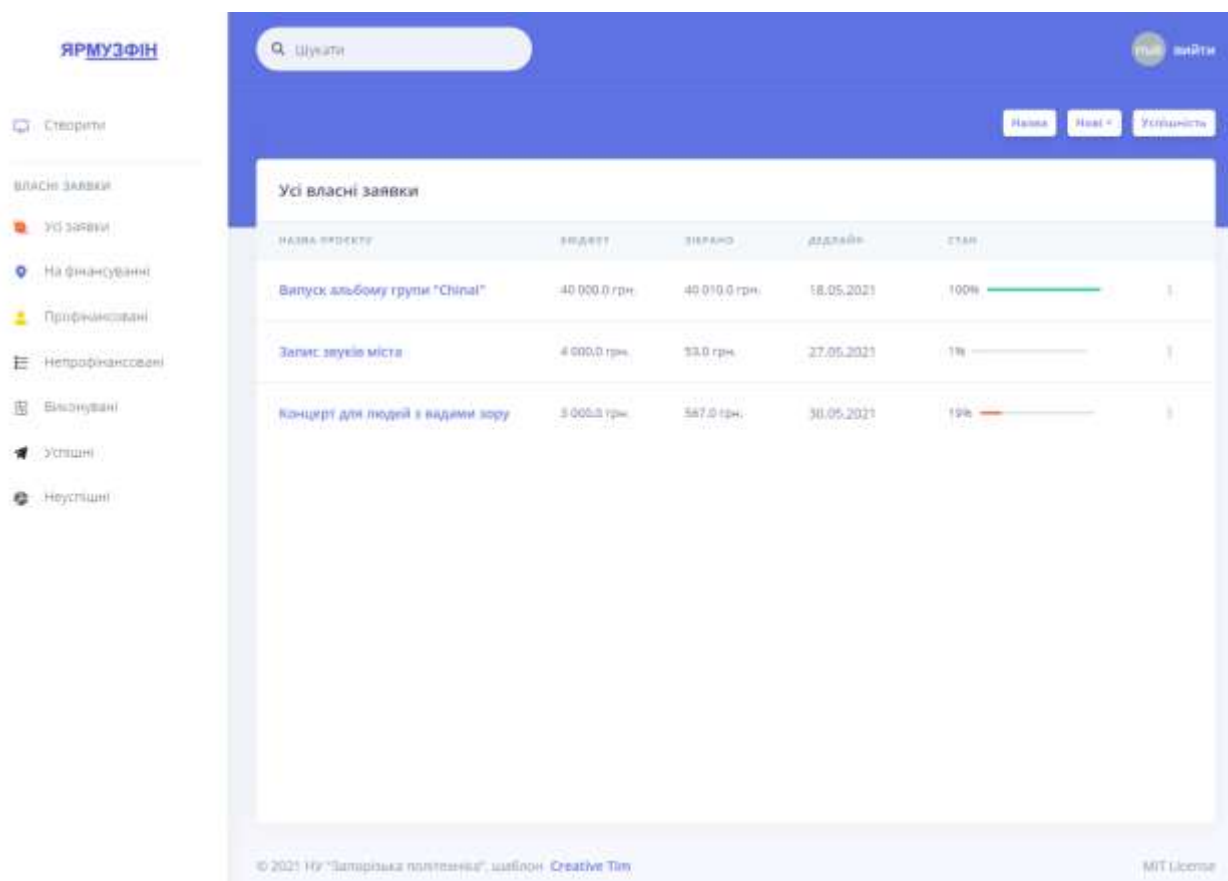


Рисунок 4.2 – Власні заявки на фінансування заявника

На даній сторінці користувачу відображається перелік усіх власних

заявок, при чому за кожною з цих відображається назва проєкту, потрібний бюджет, зібрана сума коштів, дата завершення збору коштів та відсоток зібраної суми від мети. Якщо відсоток менше 80 %, то він відображається червоним кольором. З контекстного меню проєкту доступна можливість редагувати заявку, завершити збір коштів (якщо проєкт не переведений до виконання).

Аналогічно представляються інші відображення переліків власних проєктів за допомогою публичного представлення.

Сортування можна виконати за датою створення (як у випадку рис. 4.2), за алфавітом або за успішністю (зібраним відсотком).

До пунктів меню авторизованого користувача входять:

- «Пошук», що дозволяє на відміну від пошуку в заголовку сторінки виконувати пошук по всіх музичних проєктах, але в розширеному варіанті;

- розділ «Заявки», який включає підпункти «Усі заявки» для відображення усіх раніше створених заявок на фінансування всіх користувачів, «На фінансуванні», що дозволяє переглянути тільки ті заявки, строк збору коштів на які не завершився, «Профінансовані», що дозволяє переглянути всі заявки, які отримали повне необхідне фінансування, «Непрофінансовані», що дозволяє переглянути всі заявки, які не набрали потрібну суму за відведений час, «Виконувані», що дозволяє переглянути перелік музичних проєктів, виконання яких розпочато, але не завершено, «Успішні», що дозволяє переглянути музичні проєкти, які були успішно виконано, «Неуспішні», що дозволяє переглянути музичні проєкти, які завершилися невдачею під час їх реалізації, «Найбільші», що дозволяє переглянути перелік 100 перших за сумою зібраних коштів музичних проєктів;

- «Платежі», що дозволяє переглянути всі платежі, які були виконані користувачем раніше (для неавторизованого користувач останній пункт меню відсутній, як і можливість сплатити на сторінці проєкту).

Якщо натиснути на назву будь-якої заявки на фінансування музичного проєкту, то відкриється сторінка повного опису музичного проєкту (рис. 4.3).

ЯРМУЗФІН

Пошук

Платежі

Заявки

- Усі заявки
- На фінансуванні
- Профінансовані
- Непрофінансовані
- Виконувані
- Успішні
- Неуспішні
- Найбільші

Шукати

Запис концертів вуличної музики

ЗАГАЛЬНІ ДАНІ

Опис проєкту

Вулична музика є частиною будь-якого міста, що привертає увагу до себе. Це історія з розриву "спопити мить". На згадку залишаються відео, які не несуть ніякої користі, адже запис виконаний некіслю, а мить так і лишається тільки в пам'яті. Ми хочемо запропонувати виконати якісний запис вуличної музики в українських містах. Отримані записи мають безкоштовно розповсюджуватися і привертати увагу до цих міст як їх візитівка. Для цього ми плануємо знайти звукорежисера, здатного працювати в таких умовах та орендувати потрібне обладнання. Запис буде здійснюватися у вуличних умовах. У межах цього проєкту буде реалізовано запис 40 хвилин виступів. Це перший крок у серії проєктів. Оскільки ми не маємо коштів у повній мірі, то потребуємо вашої допомоги. Допоможіть стати нашим містом трохи ближчим до кожного.

Бюджет проєкту, грн.

75 000.0 грн.

Дедлайн

24.05.2021

Стан

На фінансуванні

Внести суму

Сума фінансування

СПЛАТИТИ

ЕТАПИ ПРОЄКТУ

НОМЕР	НАЗВА	Сума	Дедлайн
1	Пошук звукорежисера	3000 грн.	31.05.2021
2	Виписування обладнання	2000 грн.	07.06.2021
3	Контракт зі звукорежисером	15000 грн.	14.06.2021
4	Передплата обладнання	10000 грн.	16.06.2021

Рисунок 4.3 – Повна інформація про проєкт

Дана сторінка містить назву проєкту, повний його опис (не міститься ні в яких табличних представленнях), потрібний бюджет проєкту, дедлайн збору коштів, стан (для виконуваного проєкту, коли кошти вже зібрано

успішно, відображається «виконуваний»), перелік етапів проєкту (якщо натиснути на назву будь-якого з етапів, то відкриється сторінка з повним описом даного етапу музичного проєкту), перелік тегів, співставлених з проєктом (можна натиснути на будь-який та перейти до сторінки відображення музичних проєктів за даним тегом), перелік коментарів. Коментарі можуть додавати тільки користувачі, які профінансували проєкт.

На правій картці розташовується відео, співставлене з проєктом, якщо воно було додано.

Куратор музичних проєктів у своєму меню також має всі пункти меню, описані для авторизованого користувача, але до них він має також окремий пункт «Кураторство» (рис. 4.4), у якому відображається перелік усіх музичних проєктів, для яких цей користувач є куратором.

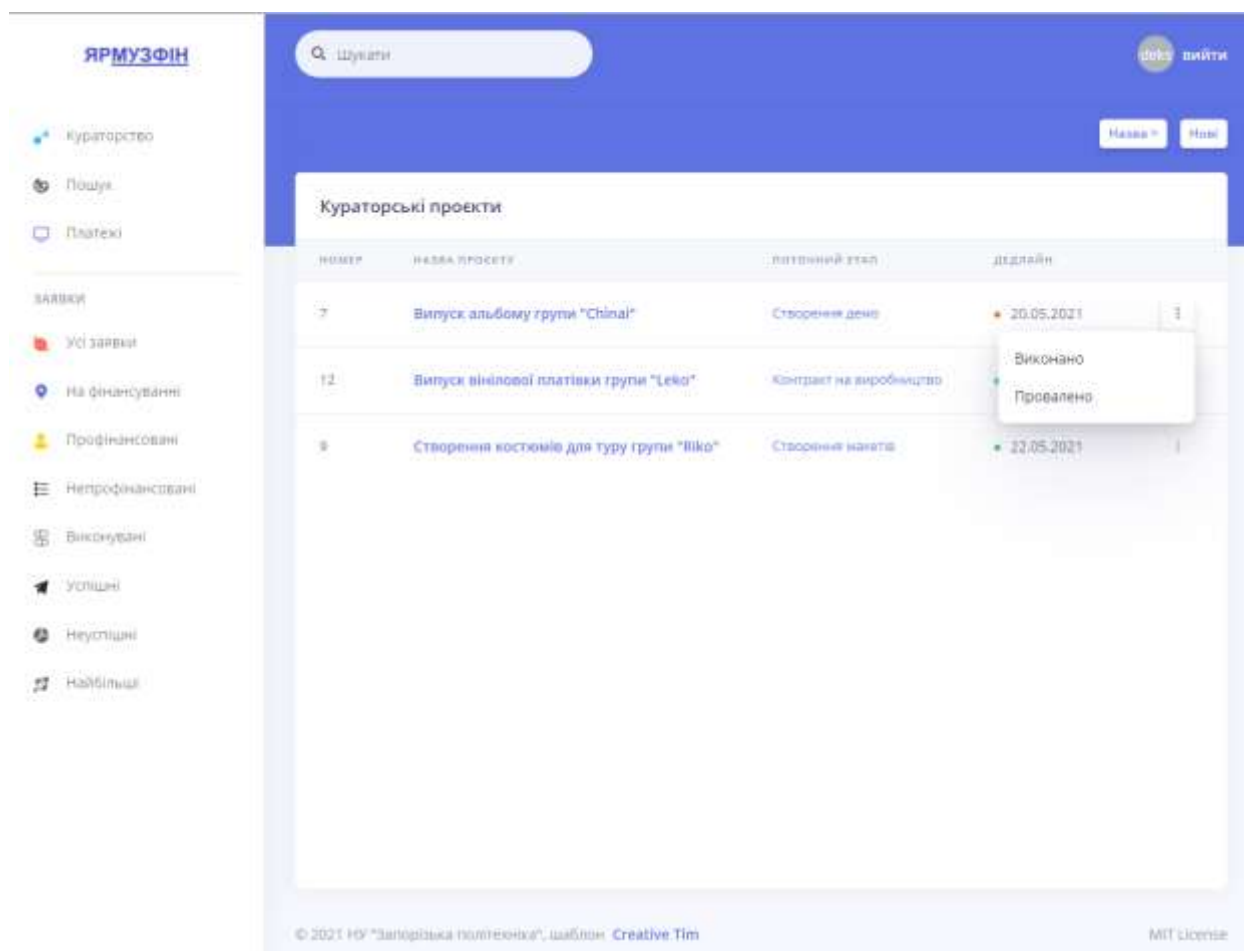


Рисунок 4.4 – Керування куратором музичними проєктами

Переліком можна керувати, сортуючи його потрібним чином. Доступні варіанти сортування за алфавітом (кнопка «Назва») та за датою створення (кнопка «Нові»). Активна кнопка в назві містить знак «>» або «<», що визначає напрямок сортування: за збільшенням або за зменшенням. Для зміни напрямку сортування потрібно натиснути на кнопку ще раз.

Куратору для роботи з проєктами доступне меню, яке містить пункти «Виконано», який потрібно обирати, якщо куратор повністю впевнений в тому, що поточний етап проєкту завершений успішно, та «Провалено», який куратор обирає в тому випадку, якщо впевнений, що даний етап не вдалося завершити достатньо якісно. У останньому випадку фінансування проєкту завершується, роботи з ним зупиняється.

Для кожного проєкту відображається окрім його назви назва поточного виконуваного етапу, за якою можна перейти до сторінки деталізації інформації про обраний етап. У стовпці «Дедлайн» відображається дата, до якої має бути завершено етап проєкту. Якщо дата пізніша за поточну, а куратор не обрав варіант дії, то біля дати розташований червоний круг. У протилежному випадку відображається зелений круг.

ВИСНОВКИ

У дипломній кваліфікаційній роботі розв'язано проблему розроблення програмного забезпечення поетапного громадського фінансування музичних проєктів для підтримки безпечної процедури фінансування:

- розглянуто принципи поетапного громадського фінансування музичних проєктів, проаналізовано основні існуючі програмні рішення, серед яких основними є Kickstarter та Patreon, їх функціональні можливості, виділено їх недоліки, виконано порівняння цих рішень з рішенням, створеним в роботі, виділено вимоги до програми;

- виконано розробку архітектури програми: виконано вибір мови програмування Python і вебфреймворку Django для реалізації програми, визначено архітектуру програмного забезпечення на основі архітектурного шаблону MVT, визначено функціональні вимоги до програми, побудовано діаграми прецедентів для роботи адміністратора, заявника, куратора, авторизованого та неавторизованого користувачів з програмою, визначено структуру даних за допомогою схеми бази даних;

- реалізовано програмне забезпечення у вигляді вебзастосунку, за результатами чого описано функціонування програми, структуру створеного рішення.

Вебзастосунок, який було розроблено в роботі, надає можливість створення заявок на фінансування музичних проєктів, керування цими заявками, створення власних внесків на фінансування музичних проєктів, які надаються проєкту поетапно, після того, як куратор проєкту підтвердив виконання попереднього етапу, і повертаються користувачу, якщо потрібну суму на фінансування не вдалося зібрати.

Усі завдання дипломної кваліфікаційної роботи повністю виконано.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Seydel, R. Crowdfunding for Musicians [Electronic resource] : How to Get Money For Your Next Project / R. Seydel. – Access mode : <https://blog.landr.com/crowdfunding-for-musicians/>.
2. Music Crowdfunding [Electronic resource]. – Access mode : <https://musiciansunion.org.uk/events-career-development/career-development/career-guides/financing-and-funding-your-work/music-crowdfunding>.
3. A Guide to Music Crowdfunding for Artists [Electronic resource]. – Access mode : <https://rotorvideos.com/blog/music-crowdfunding-artists-guide>.
4. Kickstarter [Electronic resource]. – Access mode : <https://www.kickstarter.com/>.
5. McLaughlin, M. Kickstarter Review | PCMag [Electronic resource] / M. McLaughlin, J. Minor. – Access mode : <https://www.pcmag.com/reviews/kickstarter>.
6. Best way for artists and creators to get sustainable income and connect with fans | Patreon [Electronic resource]. – Access mode : <https://www.patreon.com/>.
7. Vissers, J. Patreon Review [Electronic resource] / J. Vissers. – Access mode : <https://www.merchantmaverick.com/reviews/patreon-review/>.
8. McLaughlin, M. Patreon Review [Electronic resource] / M. McLaughlin, J. Minor. – Access mode : <https://www.pcmag.com/reviews/patreon>.
9. Murphy, C. J. Patreon Review: Worth It or Not? [Electronic resource] / C. J. Murphy. – Access mode : <https://www.musicianwithadayjob.com/patreon-review/>.
10. Building Websites with Django [Text] : Build and deploy professional websites with Python programming and the Django framework / Awanish Ranjan. – Mumbai : BPB Publications, 2021. – 242 p.

11. Web Development with Django [Text] : Learn to build modern web applications with a Python-based framework / Ben Shaw, Saurabh Badhwar, Andrew Bird, Bharath Chandra, Chris Guest. – Birmingham : Packt Publishing, 2021. – 826 p.
12. Лутц, М. Программирование на Python [Текст] : пер. с англ. / М. Лутц. – 4-е изд. – СПб. : Символ-Плюс, 2011. – 992 с.
13. Django vs Node.JS – Here's a 3 Minutes Detailed Comparison Analysis [Electronic resource]. – Access mode : <https://www.dailyrazor.com/blog/django-vs-node-js/>.
14. Difference Between MVC And MVT Patterns [Electronic resource]. – Access mode : <https://pythonistaplanet.com/difference-between-mvc-and-mvt/>.
15. LiqPay python sdk [Electronic resource]. – Access mode : <https://github.com/liqpay/sdk-python>.
16. Argon – Free and Open Source Dashboard for Bootstrap 4 @ Creative Tim [Electronic resource]. – Access mode : <https://www.creative-tim.com/product/argon-dashboard#>.

ДОДАТОК А
Технічне завдання

Вступ

Поетапне фінансування проєкту дозволить повністю контролювати процедуру реалізації музичного проєкту, що повинно підвищити впевненість спонсорів у тому, що незважаючи на специфіку взаємодії з творчими людьми, результат виконання проєкту буде досягнуто.

A.1 Підстави для розробки

Підставою для розробки було завдання на дипломну кваліфікаційну роботу за темою «Розробка програмного забезпечення поетапного громадського фінансування музичних проєктів» на основі наказу № 103 від 30 березня 2021 р. за Національним університетом «Запорізька політехніка».

A.2 Призначення розробки

Програма, що розробляється в роботі, повинна організовувати підтримку громадського фінансування музичних проєктів, надаючи можливість створення заявок на фінансування музичних проєктів, керування цими заявками, створення власних внесків на фінансування музичних проєктів, які надаються проєкту поетапно, після того, як куратор проєкту підтвердив виконання попереднього етапу, і повертаються користувачу, якщо потрібну суму на фінансування не вдалося зібрати.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Функціональні характеристики програмного забезпечення повинні включати:

- створення заявки на фінансування музичного проєкту;

- визначення етапів виконання і фінансування музичного проєкту;
- редагування створеної заявки на фінансування музичного проєкту;
- перегляд власних створених заявок на фінансування музичного проєкту;
- завершення збору коштів на музичний проєкт до початку першого етапу;
- підтвердження виконання етапу виконання музичного проєкту;
- підтвердження невиконання етапу виконання музичного проєкту;
- перерахування коштів на виконання музичного проєкту;
- відображення інформації про музичний проєкт;
- внесення коментарів на профінансований користувачем музичний проєкт;
- пошук музичних проєктів;
- відображення власних фінансувань музичних проєктів;
- перегляд сум, внесених на фінансування музичного проєкту;
- реєстрація облікових записів користувачів;
- визначення для користувача ролі заявника або куратора;
- авторизація облікового запису;
- відображення музичних проєктів, які отримали найбільше фінансування;
- відображення музичних проєктів за часом створення;
- відображення профінансованих музичних проєктів;
- відображення музичних проєктів, які знаходяться на фінансуванні і отримали його перший етап;
- відображення музичних проєктів, які не отримали фінансування;
- відображення музичних проєктів, які отримали початкове фінансування, але не були завершені.

А.3.2 Вимоги до надійності

Програма має обробляти некоректні введення користувача або забороняти можливість внесення таких даних.

А.3.3 Умови експлуатації

Умовами експлуатації програми є наявність адміністратора, який повинен забезпечити роботу вебсервера та сервера бази даних.

А.3.4 Вимоги до складу та параметрів технічних засобів

Мінімальні вимоги до серверної сторони:

- жорсткий диск з вільним простором 1 Гб;
- оперативна пам'ять обсягом 4 Гб;
- процесор з 4 ядрами і частотою 2,8 ГГц.

Для виконання вимог до клієнтської сторони потрібний встановлений сучасний браузер з доступом до мережі Інтернет.

А.4 Порядок контролю та приймання

Контроль за виконанням дипломної кваліфікаційної роботи покладається на керівника. Остаточне рішення приймається екзаменаційною комісією під час захисту роботи.

ДОДАТОК Б
Текст програми

Б.1 Текст файла `views/musproject.py`

```

from django.views.generic import ListView, UpdateView
from django.views.generic.edit import CreateView
from django.views.generic.detail import DetailView
from django.shortcuts import get_object_or_404
import datetime
from django.contrib.auth.mixins import LoginRequiredMixin
from .models import *
from .forms import *

class MusProjectsView (ListView):
    model = MusProject
    template_name = 'musproject/index.html'
    paginate_by = 10

    def get_queryset(self):
        self.ordering = self.request.GET.get('ordering')
        data = MusProject.objects.all()
        if self.ordering == 0:
            data = data.order_by('title')
        elif self.ordering == 1:
            data = data.order_by('-created')
        elif self.ordering == 2:
            data = data.order_by('created')
        elif self.ordering == 3:
            data = data.order_by('title')
        elif self.ordering == 4:
            data = data.order_by('-done')
        elif self.ordering == 5:
            data = data.order_by('done')
        return data

    def get_context_data(self, **kwargs):
        data = super(MusProjectsView,
self).get_context_data(**kwargs)
        data ['ordering'] = self.ordering
        return data

class OwnMusProjectView(LoginRequiredMixin, DetailView):
    model = MusProject
    template_name = 'musproject/odetail.html'

```

```

        def get_context_data(self, **kwargs):
            data = super(MusProjectView,
self).get_context_data(**kwargs)
            data['stages'] =
MusProjectStage.objects.filter(musproject__id
self.kwarg.get('pk')).order_by('order')
            data['payoffs'] =
Payoff.objects.filter(musproject__id
self.kwarg.get('pk')).order_by('created')
            data['comments'] =
Comment.objects.filter(musproject__id
self.kwarg.get('pk')).order_by('created')
            return data

```

```

class MusProjectView(DetailView):
    model = MusProject
    template_name = 'musproject/detail.html'

    def get_context_data(self, **kwargs):
        data = super(MusProjectView,
self).get_context_data(**kwargs)
        data['stages'] =
MusProjectStage.objects.filter(musproject__id
self.kwarg.get('pk')).order_by('order')
        ps = Payoff.objects.filter(musproject__id
self.kwarg.get('pk'))
        users = []
        for p in ps:
            if p.user not in users:
                users.append(p)
        data['payoffs'] = users
        data['comments'] =
Comment.objects.filter(musproject__id
self.kwarg.get('pk')).order_by('created')
        return data

```

```

class OwnMusProjectsView (LoginRequiredMixin, ListView):
    model = MusProject
    template_name = 'musproject/own.html'
    paginate_by = 10

    def get_queryset(self):
        data = MusProject.objects.all()
        self.ordering = self.request.GET.get('ordering')

```

```

    authuser = self.request.user
    data = data.filter(author = authuser)
    if self.ordering == 0:
        data = data.order_by('title')
    elif self.ordering == 1:
        data = data.order_by('-created')
    elif self.ordering == 2:
        data = data.order_by('created')
    elif self.ordering == 3:
        data = data.order_by('title')
    elif self.ordering == 4:
        data = data.order_by('-done')
    elif self.ordering == 5:
        data = data.order_by('done')
    return data

    def get_context_data(self, **kwargs):
        data = super(OwnMusProjectsView,
self).get_context_data(**kwargs)
        data ['ordering'] = self.ordering
    return data

class OwnSuccessMusProjectsView (LoginRequiredMixin,
ListView):
    model = MusProject
    template_name = 'musproject/ownsuccess.html'

    def get_queryset(self):
        data = MusProject.objects.all()
        authuser = self.request.user
        data = data.filter(author = authuser)
        data = data.filter(status = 1)
        data.order_by('-created')
        return data

class OwnFailedMusProjectsView (LoginRequiredMixin,
ListView):
    model = MusProject
    template_name = 'musproject/ownfailed.html'

    def get_queryset(self):
        data = MusProject.objects.all()
        authuser = self.request.user
        data = data.filter(author = authuser)

```

```

        data = data.filter(status = 2)
        data.order_by('-created')
        return data

```

```

class      OwnUndoneMusProjectsView      (LoginRequiredMixin,
ListView):
    model = MusProject
    template_name = 'musproject/ownundone.html'

    def get_queryset(self):
        data = MusProject.objects.all()
        authuser = self.request.user
        data = data.filter(author = authuser)
        data = data.filter(status = 4)
        data.order_by('-created')
        return data

```

```

class      OwnDoneMusProjectsView      (LoginRequiredMixin,
ListView):
    model = MusProject
    template_name = 'musproject/owndone.html'

    def get_queryset(self):
        data = MusProject.objects.all()
        authuser = self.request.user
        data = data.filter(author = authuser)
        data = data.filter(status = 5)
        data.order_by('-created')
        return data

```

```

class      OwnInprocessProjectsView      (LoginRequiredMixin,
ListView):
    model = MusProject
    template_name = 'musproject/owninproc.html'

    def get_queryset(self):
        data = MusProject.objects.all()
        authuser = self.request.user
        data = data.filter(author = authuser)
        data = data.filter(status = 3)
        data.order_by('-created')
        return data

```

```

class OwnFinanceProjectsView(LoginRequiredMixin,
ListView):
    model = MusProject
    template_name = 'musproject/ownfin.html'

    def get_queryset(self):
        data = MusProject.objects.all()
        authuser = self.request.user
        data = data.filter(author = authuser)
        data = data.filter(status = 0)
        data.order_by('-done')
        return data

class CuratorMusProjectsView(LoginRequiredMixin,
ListView):
    model = MusProject
    template_name = 'musproject/curator.html'
    paginate_by = 10

    def get_queryset(self):
        data = MusProject.objects.all()
        self.ordering = self.request.GET.get('ordering')
        authuser = self.request.user
        data = data.filter(curator = authuser)
        if self.ordering == 0:
            data = data.order_by('title')
        elif self.ordering == 1:
            data = data.order_by('-created')
        elif self.ordering == 2:
            data = data.order_by('created')
        elif self.ordering == 3:
            data = data.order_by('title')
        elif self.ordering == 4:
            data = data.order_by('-done')
        elif self.ordering == 5:
            data = data.order_by('done')
        return data

    def get_context_data(self, **kwargs):
        data = super(CuratorMusProjectsView,
self).get_context_data(**kwargs)
        data['ordering'] = self.ordering
        return data

```

```

class SearchMusProjectsView (ListView):
    model = MusProject
    template_name = 'musproject/search.html'
    paginate_by = 10

    def get_queryset(self):
        data = MusProject.objects.all()
        self.ordering = self.request.GET.get('ordering')
        self.searchv = self.request.GET.get('title')
        self.created1 = self.request.GET.get('data1')
        self.created2 = self.request.GET.get('data2')
        self.plus = self.request.GET.get('plus')
        self.costs1 = self.request.GET.get('sum1')
        self.costs2 = self.request.GET.get('sum2')
        self.gcosts1 = self.request.GET.get('gsum1')
        self.gcosts2 = self.request.GET.get('gsum2')
        if self.searchv:
            data = data.filter(title__contains =
self.searchv)
        if self.plus:
            data = data.filter(description__contains =
self.searchv)
        if self.costs1:
            data = data.filter(costs__gte = self.costs1)
        if self.costs2:
            data = data.filter(costs__lte = self.costs2)
        if self.gcosts1:
            data = data.filter(gathcosts__gte =
self.gcosts1)
        if self.gcosts2:
            data = data.filter(gathcosts__lte =
self.gcosts2)
        if self.created1:
            data = data.filter(created__gte =
self.created1)
        if self.created2:
            data = data.filter(created__lte =
self.created2)
        if self.ordering == 0:
            data = data.order_by('title')
        elif self.ordering == 1:
            data = data.order_by('-created')
        elif self.ordering == 2:
            data = data.order_by('created')
        elif self.ordering == 3:
            data = data.order_by('title')

```

```

        elif self.ordering == 4:
            data = data.order_by('-done')
        elif self.ordering == 5:
            data = data.order_by('done')
        return data

    def get_context_data(self, **kwargs):
        data = super(SearchMusProjectsView,
self).get_context_data(**kwargs)
        data ['ordering'] = self.ordering
        data ['title'] = self.searchv
        data ['data1'] = self.created1
        data ['data2'] = self.created2
        data ['sum1'] = self.costs1
        data ['sum2'] = self.costs2
        data ['gsum1'] = self.gcosts1
        data ['gsum2'] = self.gcosts2
        return data

class MusProjectUpdateView(LoginRequiredMixin, UpdateView):
    template_name = 'musproject/update.html'
    form_class = MusProjectUpdateForm

class MusProjectFinishView(LoginRequiredMixin, UpdateView):
    template_name = 'mp/update.html'
    form_class = MusProjectUpdateForm

    def form_valid(self, form):
        form = form.save(commit=False)
        form.successend()
        form.save()
        self.object = form
        return super().form_valid(form)

class MusProjectUnFinishView(LoginRequiredMixin,
UpdateView):
    template_name = 'musproject/update.html'
    form_class = MusProjectUpdateForm

    def form_valid(self, form):
        form = form.save(commit=False)
        form.stop()
        form.save()

```

```

        self.object = form
        return super().form_valid(form)

class MusProjectStartView(LoginRequiredMixin, UpdateView):
    template_name = 'musproject/update.html'
    form_class = MusProjectUpdateForm

    def form_valid(self, form):
        form = form.save(commit=False)
        form.start()
        form.save()
        self.object = form
        return super().form_valid(form)

class TagMusProjectsView (ListView):
    model = MusProject
    template_name = 'musproject/tag.html'
    paginate_by = 10

    def get_queryset(self):
        data = MusProject.objects.all()
        self.tag = self.request.GET.get('tag')
        obj = get_object_or_404(Tag, name = self.tag)
        self.tagid = obj.id
        if self.tag:
            data = data.filter(tags__in = [self.tagid])
        return data

    def get_context_data(self, **kwargs):
        data = super(TagMusProjectsView,
self).get_context_data(**kwargs)
        data ['tag'] = self.tag
        return data

class MaxFinMusProjectsView (ListView):
    model = MusProject
    template_name = 'musproject/maxfinance.html'
    paginate_by = 10

    def get_queryset(self):
        data = MusProject.objects.all()[0:100]
        data = data.order_by('gathcosts')
        for d in data:

```

```

        if d.gathcosts < d.costs:
            data.remove(d)
    return data

class SuccessMusProjectsView (ListView):
    model = MusProject
    template_name = 'musproject/success.html'
    paginate_by = 10

    def get_queryset(self):
        data = MusProject.objects.all()
        data = data.order_by('gathcosts')
        for d in data:
            if d.gathcosts < d.costs:
                data.remove(d)
        return data

class FailedMusProjectsView (ListView):
    model = MusProject
    template_name = 'musproject/failed.html'
    paginate_by = 10

    def get_queryset(self):
        data = MusProject.objects.all()
        data = data.order_by('gathcosts')
        for d in data:
            if d.gathcosts >= d.costs:
                data.remove(d)
        return data

class UndoneMusProjectsView (ListView):
    model = MusProject
    template_name = 'musproject/undone.html'
    paginate_by = 10

    def get_queryset(self):
        data = MusProject.objects.all()
        data = data.order_by('gathcosts')
        for d in data:
            if d.status != 4 and d.status:
                data.remove(d)
        return data

```

```

class DoneMusProjectsView (ListView):
    model = MusProject
    template_name = 'musproject/done.html'
    paginate_by = 10

    def get_queryset(self):
        data = MusProject.objects.all()
        data = data.order_by('-created')
        for d in data:
            if d.status != 5 and d.status:
                data.remove(d)
        return data

class InprocessMusProjectsView (ListView):
    model = MusProject
    template_name = 'musproject/inprocess.html'
    paginate_by = 10

    def get_queryset(self):
        data = MusProject.objects.all()
        data = data.order_by('')
        for d in data:
            if d.status != 3:
                data.remove(d)
        return data

class MusProjectCreateView(CreateView):
    template_name = 'musproject/create.html'
    form_class = MusProjectCreateForm

    def form_valid(self, form):
        form = form.save(commit=False)
        form.created = datetime.datetime.now()
        user = self.request.user
        userobj = get_object_or_404(User, user=user)
        form.author = userobj
        form.save()

    def get_form_kwargs(self, *args, **kwargs):
        data = super(MusProjectCreateView,
self).get_form_kwargs(*args, **kwargs)
        data ['tags'] = Tag.objects.all()
        return data

```

```

class SearchOwnMusProjectsView (ListView):
    model = MusProject
    template_name = 'musproject/osearch.html'
    paginate_by = 10

    def get_queryset(self):
        authuser = self.request.user
        data = MusProject.objects.filter(author = authuser)
        self.searchv = self.request.GET.get('title')
        if self.searchv:
            data = data.filter(title__contains =
self.searchv)
        return data

    def get_context_data(self, **kwargs):
        data = super(SearchMusProjectsView,
self).get_context_data(**kwargs)
        data ['title'] = self.searchv
        return data

```

Б.2 Текст файла models.py

```

from django.db import models
from django.urls import reverse

class MusProject (models.Model):
    title = models.CharField(max_length=100)
    description = models.TextField()
    created = models.DateField()
    deadline = models.DateField()
    costs = models.FloatField()
    gathcosts = models.FloatField ()
    status = models.SmallIntegerField()
    video = models.CharField(max_length=255)
    author = models.ForeignKey(User,
on_delete=models.CASCADE)
    curator = models.ForeignKey(User,
on_delete=models.CASCADE)
    tags = models.ManyToManyField(Tag)

    @property
    def done(self):
        res = self.gathcosts / self.costs * 100
        return res

```

```

def __str__(self):
    return self.title

class Meta:
    ordering = ['-created']

def get_absolute_url(self):
    return reverse('musproject.views.details',
args=[str(self.id)])

def endfinance(self):
    if self.gathcosts > self.costs:
        self.status = 1
    else:
        self.status = 2
    if self.status != 1 and self.status != 2:
        return False
    else:
        return True

def stopfinance(self):
    self.status = 2
    if self.status != 2:
        return False
    else:
        return True

def start(self):
    self.status = 3
    if self.status != 3:
        return False
    else:
        return True

def successend(self):
    self.status = 5
    if self.status != 5:
        return False
    else:
        return True

def stop(self):
    self.status = 4
    if self.status != 4:
        return False
    else:
        return True

```

```

class MusProjectStage (models.Model):
    title = models.CharField(max_length=100)
    costs = models.FloatField ()
    status = models.SmallIntegerField()
    deadline = models.DateField()
    description = models.TextField()
    order = models.SmallIntegerField()
    musproject = models.ForeignKey (MusProject,
on_delete=models.CASCADE)

    def __str__(self):
        return self.title

    class Meta:
        ordering = ['deadline']

    def get_absolute_url(self):
        return reverse('mpstage.views.details',
args=[str(self.id)])

    def start(self):
        self.status = 1
        if self.status != 1:
            return False
        else:
            return True

    def finance(self):
        self.status = 2
        if self.status != 2:
            return False
        else:
            return True

    def successend(self):
        self.status = 3
        if self.status != 3:
            return False
        else:
            return True

    def stop(self):
        self.status = 4
        if self.status != 4:
            return False
        else:

```

```

        return True

class Payoff (models.Model):
    created = models.DateField()
    costs = models.FloatField ()
    user = models.ForeignKey(User,
on_delete=models.CASCADE)
    musproject = models.ForeignKey (MusProject,
on_delete=models.CASCADE)
    status = models.SmallIntegerField()
    banktransaction = models.CharField(max_length=20)
    account = models.CharField(max_length=20)

    def __str__(self):
        return self.banktransaction + ' - ' + self.account

    class Meta:
        ordering = ['-created']

    def get_absolute_url(self):
        return reverse('payoff.views.details',
args=[str(self.id)])

class Tag (models.Model):
    name = models.CharField(max_length=20)

    def __str__(self):
        return self.name

class Comment (models.Model):
    text = models.TextField()
    created = models.DateTimeField()
    user = models.ForeignKey(User,
on_delete=models.CASCADE)
    musproject = models.ForeignKey (MusProject,
on_delete=models.CASCADE)

    def __str__(self):
        return self.text

    class Meta:
        ordering = ['-created']

```

```
def get_absolute_url(self):  
    return reverse('comment.views.details',  
args=[str(self.id)])
```

ДОДАТОК В
Слайди презентації



Рисунок В.1 – Слайд № 1

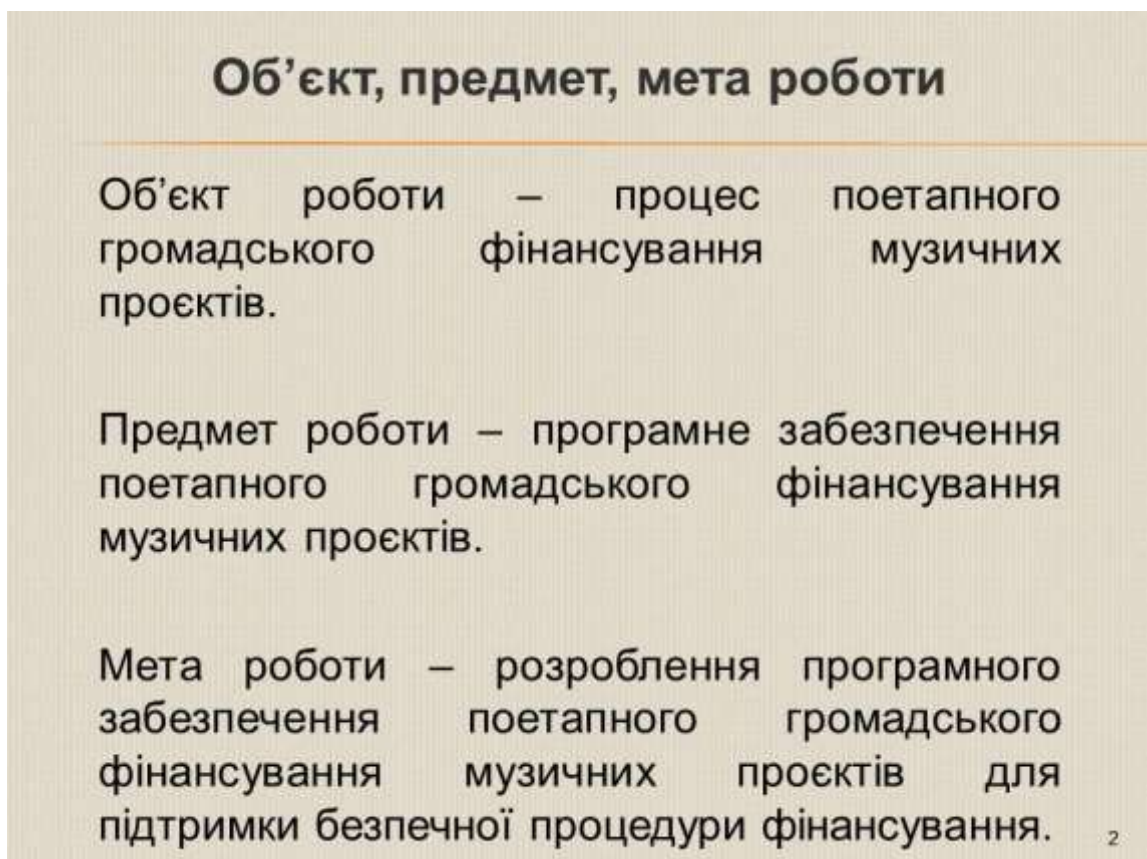


Рисунок В.2 – Слайд № 2

Завдання роботи

- ✓ розробка технічного завдання і визначення вимог до програми;
- ✓ розробка архітектури програми;
- ✓ розробка програми і тестування програми.

3

Рисунок В.3 – Слайд № 3

Порівняння програмних аналогів

Параметр	Kickstarter	Patreon	Рішення, що розробляється
Основна одиниця	Проект	Автор	Проект
Основна концепція збору і використання коштів	Все або нічого	Підписка	Етапи
Підтримка музичних проектів	Є	Є	Є
Платежі за перерахування коштів	Так	Так	Ні

4

Рисунок В.4 – Слайд № 4

Порівняння технологій розробки

Параметр	Django	Node_JS
Мова програмування	Python	JavaScript
Проблеми з безпекою	Ні	Можливі
Вбудовані шаблони	Так	Ні
Швидкість розробки	Так	Ні

5

Рисунок В.5 – Слайд № 5

Архітектурний шаблон MVT



6

Рисунок В.6 – Слайд № 6

Діаграми прецедентів адміністратора, куратора та заявника

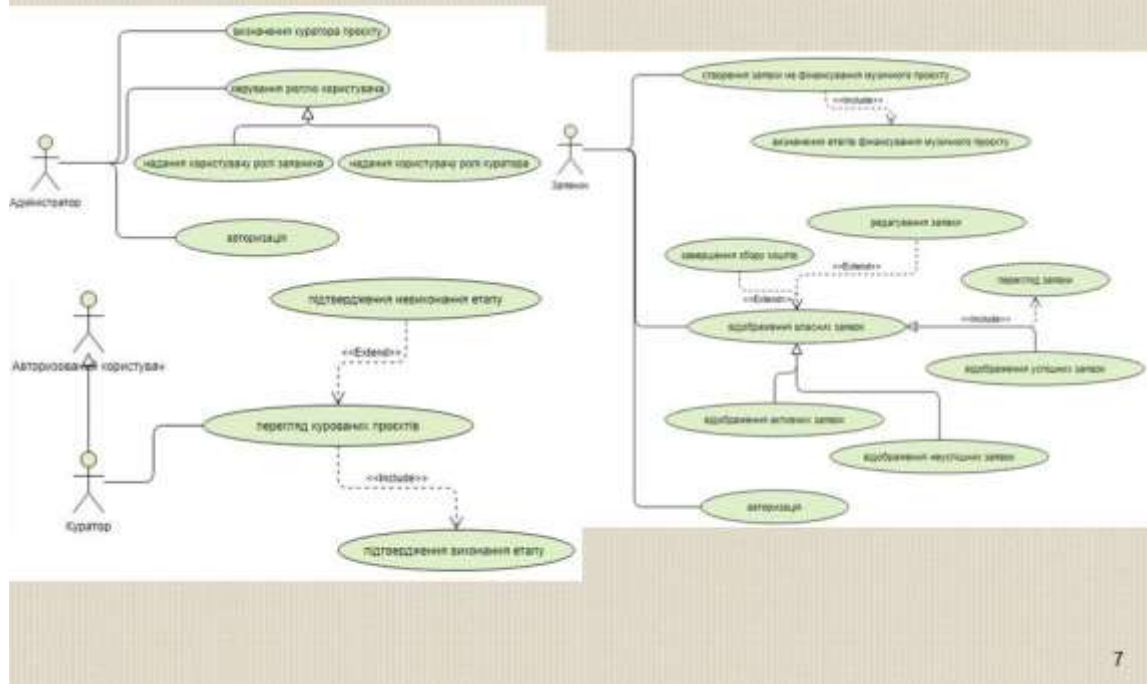


Рисунок В.7 – Слайд № 7

Діаграми прецедентів авторизованого і неавторизованого користувача

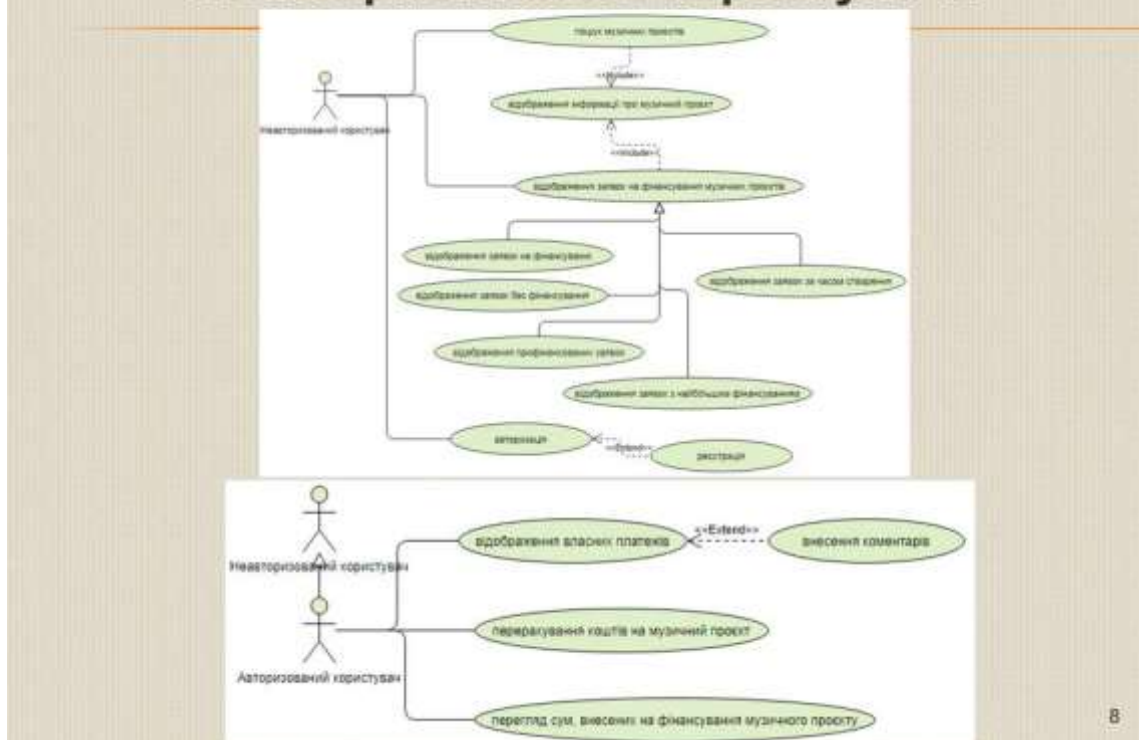
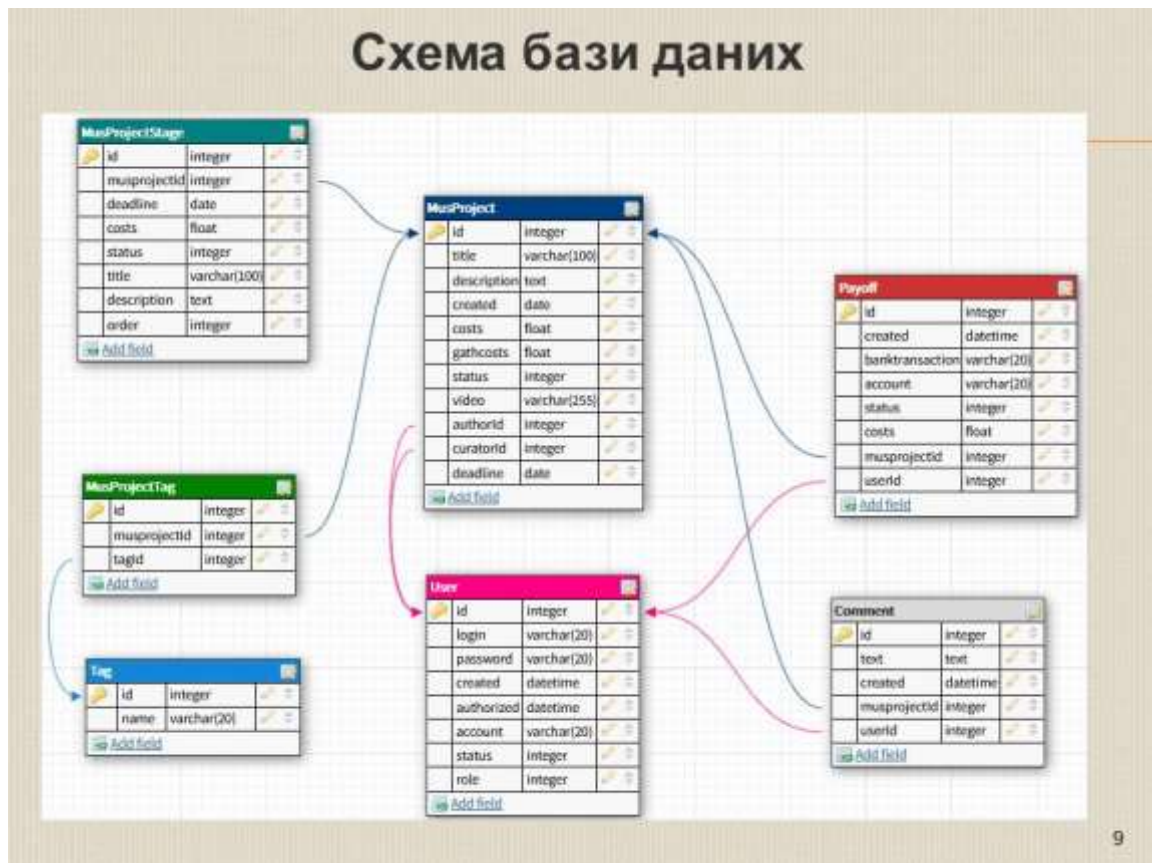


Рисунок В.8 – Слайд № 8



9

Рисунок В.9 – Слайд № 9



10

Рисунок В.10 – Слайд № 10



Рисунок В.11 – Слайд № 11

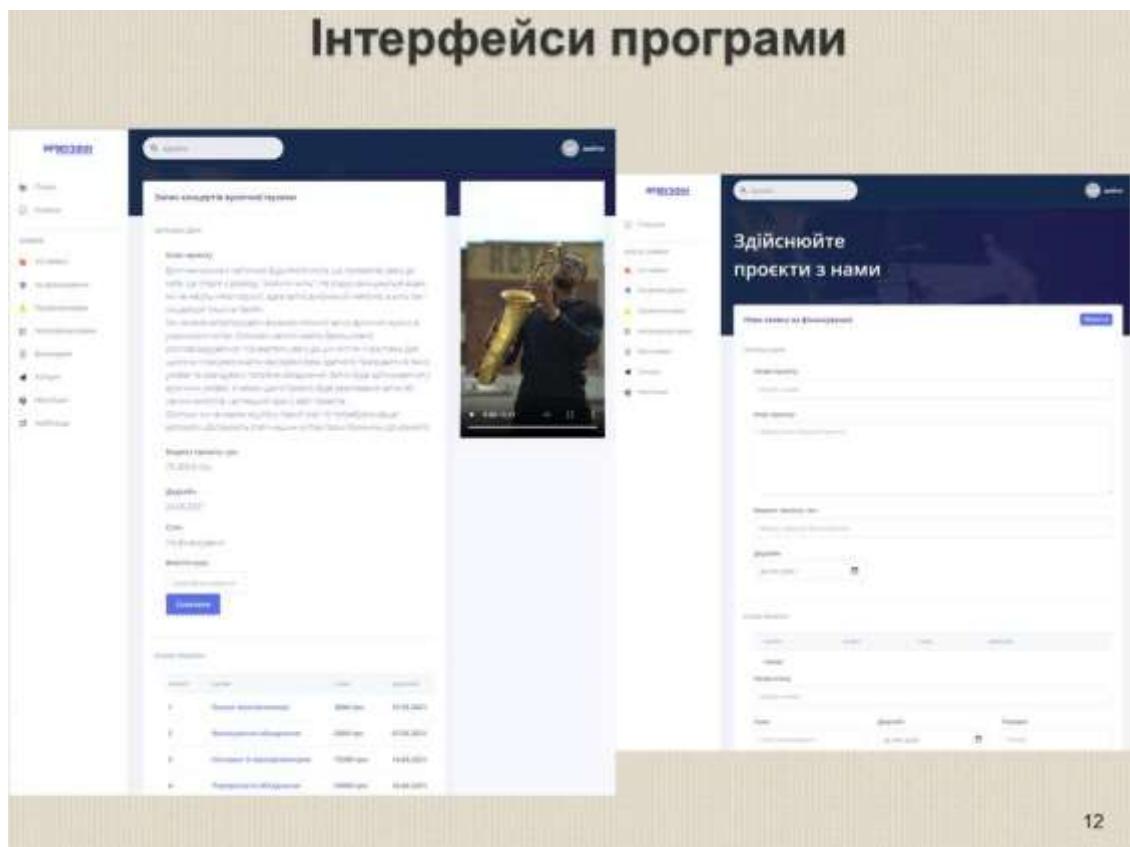


Рисунок В.12 – Слайд № 12

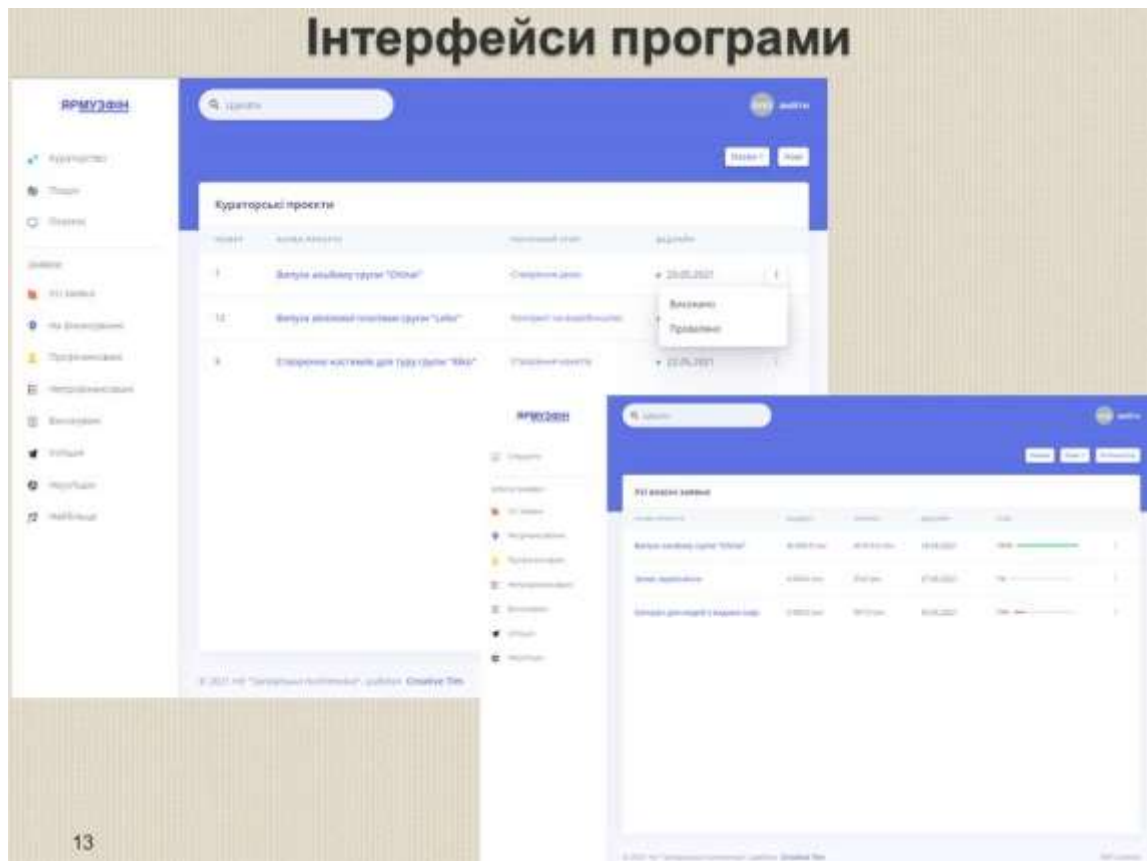


Рисунок В.13 – Слайд № 13

Висновки

У дипломній кваліфікаційній роботі розв'язано проблему розроблення програмного забезпечення поетапного громадського фінансування музичних проєктів для підтримки безпечної процедури фінансування:

– розглянуто принципи поетапного громадського фінансування музичних проєктів, проаналізовано основні існуючі програмні рішення, серед яких основними є Kickstarter та Patreon, їх функціональні можливості, виділено їх недоліки, виконано порівняння цих рішень з рішенням, створеним в роботі, виділено вимоги до програми;

– виконано розробку архітектури програми: виконано вибір мови програмування Python і вебфреймворку Django для реалізації програми, визначено архітектурну програмуного забезпечення на основі архітектурного шаблону MVT, визначено функціональні вимоги до програми, побудовано діаграми прецедентів для роботи адміністратора, заявника, куратора, авторизованого та неавторизованого користувачів з програмою, визначено структуру даних за допомогою схеми бази даних;

– реалізовано програмне забезпечення у вигляді вебзастосунку, за результатами чого описано функціонування програми, структуру створеного рішення.

Усі завдання дипломної кваліфікаційної роботи повністю виконано.

Рисунок В.14 – Слайд № 14