

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Розробка платформи для подяк стрімерам на Twitch»

Виконав студент групи ІПЗ-20-4
ГІССА Дмитро Олексійович

Керівник роботи:
САВІЦЬКИЙ Роман Святославович

Рецензент:
ЧИЖМОТРА Олексій Володимирович

Житомир – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна Вакалюк

«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ГІССА Дмитро Олексійович**

Керівник роботи: **САВІЦЬКИЙ Роман Святославович**

Тема роботи: **«Розробка платформи для подяк стрімерам на Twitch»**,
затверджена Наказом закладу вищої освіти від «23» лютого 2024 р., №74/с

Вихідні дані для роботи: Об'єктом дослідження є використання інформаційних технологій для спрощення збору донатів на різні цілі та покращення користувацького досвіду стрімерів.

Предметом дослідження є використання сучасних веборієнтованих технологій розробки для створення вебплатформи, що спрощує та покращує процес збору коштів на певні цілі.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Савіцький Р.С.	27.02.2024р.	27.02.2024р.
2	Савіцький Р.С.	27.02.2024р.	27.02.2024р.
3	Савіцький Р.С.	27.02.2024р.	27.02.2024р.

РЕФЕРАТ

Випускна робота бакалавра складається з програмного комплексу для платформи подяк і донатів стрімерам на Twitch, а також пояснювальної записки.

Пояснювальна записка до випускної роботи містить текстову частину, викладену на 58 сторінках друкованого тексту, має 8 сторінок додатків. Список використаних джерел містить 22 найменування і займає 2 сторінки. В роботі наведено 47 ілюстрацій та 17 таблиць. Загальний обсяг роботи – 75 сторінок.

Метою роботи є розробка платформи для подяк стрімерам на Twitch з використанням засобів Python, Django, HTML, CSS, JavaScript й бази даних PostgreSQL.

У роботі було визначено основні проблеми, які повинна вирішувати платформа, та сформовано і обґрунтовано завдання на розробку системи. Було обрано і розглянуто аналогічні продукти з метою визначення їхніх переваг і недоліків. Також було створено і описано алгоритми роботи платформи та сформовано її загальну архітектуру. Окрему увагу приділено питанням інтеграції з API Twitch для забезпечення коректної взаємодії платформи зі стрімінговим сервісом. Було реалізовано підтримку інтеграції зі стрімінговими програмами, такими як OBS, що дозволяє стрімерам легко налаштовувати та використовувати платформу у своїх трансляціях. Також було створено тестові сценарії для перевірки функціональності та надійності системи.

КЛЮЧОВІ СЛОВА: БАЗА ДАНИХ, ПЛАТФОРМА, ТРАНСЛЯЦІЯ, СТІМЕР, ДОНАТ, TWITCH, ЗБІР.

					ІПЗ.КР.Б – 121 – 24 – ПЗ							
Змн.	Арк.	№ докум.	Підпис	Дата								
Розроб.		Гісса Д.О.			Платформа для подяк стрімерам на Twitch			Літ.	Арк.	Аркушів		
Керівник		Савіцький Р.С.								3	75	
								Житомирська політехніка, група ІПЗ-20-4				
Рецензент		Чижмоторя О.В.										
Зав. каф.		Вакалюк Т.А.										

ABSTRACT

The bachelor's thesis consists of a software package for a platform of gratitude and donations to streamers on Twitch, as well as an explanatory note.

The explanatory note to the graduation thesis contains the text part, set out on 58 pages of printed text, and has 8 pages of appendices. The list of references includes 22 titles and occupies 2 pages. The work contains 47 illustrations and 17 tables. The total volume of the work is 75 pages.

The purpose of the work is to develop a platform for thanking streamers on Twitch using Python, Django, HTML, CSS, JavaScript, and the PostgreSQL database.

The paper identifies the main problems that the platform should solve, and formulates and substantiates the task of developing the system. Similar products were selected and reviewed to determine their advantages and disadvantages. The algorithms of the platform were also created and described, and its general architecture was formed. Particular attention was paid to the integration with the Twitch API to ensure the correct interaction of the platform with the streaming service. Support for integration with streaming programs such as OBS was implemented, allowing streamers to easily set up and use the platform in their broadcasts. We also created test scenarios to check the functionality and reliability of the system.

KEYWORDS: DATABASE, PLATFORM, BROADCASTING, STREAMER, DONATION, TWITCH, MONEY RAISING.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	7
ВСТУП	8
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ ПЛАТФОРМИ ДЛЯ ПОДЯК СТІМЕРАМ НА TWITCH	10
1.1 Постановка задачі	10
1.2 Аналіз аналогів програмного продукту	11
1.3 Вибір архітектури веборієнтованої платформи для подяк стрімерам на Twitch	15
1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	17
Висновки до першого розділу	19
РОЗДІЛ 2. ПРОЄКТУВАННЯ ПЛАТФОРМИ ДЛЯ ПОДЯК СТІМЕРАМ НА TWITCH	21
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи	21
2.2 Розробка бази даних системи	27
2.3 Проектування та реалізація алгоритмів роботи системи	32
2.4 Реалізація платформи для подяк стрімерам на Twitch	35
Висновки до розділу 2	43
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ПЛАТФОРМОЮ ДЛЯ ПОДЯК СТІМЕРАМ НА TWITCH	45
3.1 Порядок встановлення та налаштування параметрів платформи	45
3.2 Структура інтерфейсу платформи для подяк стрімерам на Twitch	46
3.3 Тестування роботи програмного продукту	58
Висновки до третього розділу	61
ВИСНОВКИ	63

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

65

ДОДАТКИ

67

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	
						6

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CRUD – create, read, update, delete (з англ. створити, прочитати, оновити, видалити)

CSS – це спеціальна мова стилю сторінок, що використовується для опису їхнього зовнішнього вигляду

HTML – стандартизована мова розмітки документів для перегляду вебсторінок у браузері

MVC – архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення

SPA – single page application (з англ. односторінковий вебдодаток)

UI – дизайн інтерфейсу користувача

БД – база даних

ПІН – індивідуальний податковий номер

РНОКПП – реєстраційний номер облікової картки платника податків

СКБД – система керування базами даних

ВСТУП

Актуальність теми. Сучасні стрімінгові платформи, такі як Twitch [1], набирають величезну популярність, об'єднуючи мільйони користувачів з усього світу. Одна з ключових складових цього процесу – підтримка глядачів, зокрема фінансові подяки стрімерам за їхній контент.

Створення спеціалізованої платформи для подяк стрімерам є надзвичайно актуальним завданням. Це забезпечує зручні та ефективні механізми взаємодії між стрімерами та їхньою аудиторією, сприяючи розвитку спільноти та підтримці авторів контенту.

Платформи для подяк допомагають стрімерам монетизувати свою діяльність, що дозволяє їм продовжувати створювати якісний контент та інвестувати в свій розвиток. Така підтримка з боку глядачів також сприяє зміцненню взаємозв'язків всередині спільноти та підвищенню лояльності до платформи.

Метою даної роботи є розробка платформи для подяк стрімерам на Twitch з використанням технологій Python [2], Django [3], HTML [4], CSS [5], JavaScript [6] й PostgreSQL [7], що забезпечить глядачам легкий і швидкий спосіб виразити свою підтримку, а стрімерам – ефективні інструменти для управління отриманими подяками. Платформа повинна включати зручний та інтуїтивно зрозумілий інтерфейс, який би дозволяв користувачам без труднощів здійснювати фінансові транзакції.

Крім того, важливим аспектом є інтеграція з стрімінговими програмами, такими як OBS (Open Broadcaster Software) [8], для автоматичного відображення подяк у реальному часі, що підвищить інтерактивність та залученість глядачів.

Також платіжна система зобов'язана підтримувати декілька валют, але в першу чергу гривню, оскільки основа цільова аудиторія це саме українські користувачі.

Отже, вищезазначена мета потребує виконання таких завдань:

- провести аналіз вимог до платформи та дослідити потреби стрімерів і глядачів;
- проаналізувати переваги і недоліки аналогічних програмних продуктів;
- визначити, які технології краще використовувати для реалізації платформи;
- розробити архітектуру системи, враховуючи обрані технології;
- реалізувати серверну частину платформи, забезпечивши необхідну функціональність для обробки подяк;
- створити базу даних для зберігання інформації про користувачів та подяки;
- розробити клієнтську частину платформи з інтуїтивно зрозумілим інтерфейсом;
- інтегрувати платіжну систему, яка забезпечує підтримку гривні;
- забезпечити можливість інтеграції зі стрімінговими програмами, такими як OBS, для автоматичного відображення подяк у реальному часі;
- провести тестування платформи для виявлення та усунення можливих помилок;
- забезпечити безпеку платформи, включаючи захист даних користувачів та фінансових транзакцій;
- провести закриті бета-тестування платформи із залученням декількох реальних стрімерів для збору відгуків;
- повноцінно запустити платформу та забезпечити її подальшу підтримку та оновлення.

Об’єктом дослідження є використання інформаційних технологій для спрощення збору донатів на різні цілі та покращення користувацького досвіду стрімерів.

Предметом дослідження є використання сучасних веборієнтованих технологій розробки для створення вебплатформи, що спрощує та покращує процес збору коштів на певні цілі.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ ПЛАТФОРМИ ДЛЯ ПОДЯК СТІМЕРАМ НА TWITCH

1.1 Постановка задачі

Основна мета роботи полягає у створенні платформи для подяк стрімерам на Twitch, спрямованої на забезпечення зручності, ефективності та інтерактивності взаємодії між стрімерами та їхньою аудиторією. Така платформа має бути корисною як для глядачів, так і для стрімерів, забезпечуючи зручні інструменти для вираження та отримання подяк.

Для стрімерів платформа надасть дієві інструменти для управління отриманими подяками. Вони зможуть легко сортувати подяки, відображати їх на екрані під час стріму, а також відповідати на них у реальному часі. Це дозволить стрімерам зосередитись на створенні контенту, водночас підтримуючи тісний контакт зі своєю аудиторією.

Розробка зручного інтерфейсу є однією з основних задач проекту. Користувацький інтерфейс має бути інтуїтивно зрозумілим для обох сторін – і для глядачів, і для стрімерів. Важливо створити середовище, де кожен користувач зможе легко знайти необхідні функції та швидко освоїтися.

Інтеграція з платформою Twitch та популярними стрімінговими програмами, такими як OBS, є критично важливою. Це дозволить автоматизувати процеси реєстрації, автентифікації та обміну даними, забезпечуючи чудовий досвід користувачів. А інтеграція з OBS дозволить автоматично відображати подяки на екрані під час стріму, що підвищить рівень взаємодії між стрімером та його аудиторією.

Однією з ключових особливостей платформи має бути використання платіжної системи, яка підтримує різні валюти, але з пріоритетною підтримкою гривні. Це дозволить глядачам здійснювати пожертви у зручний для них спосіб, незалежно від їх місця проживання.

В результаті виконання всіх зазначених завдань буде створена веборієнтована платформа для подяк стрімерам на Twitch. Вона буде складатися з серверної частини, що відповідає за обробку бізнес-логіки, управління базою даних та зберігання інформації, а також клієнтської частини з простим і інтуїтивно зрозумілим інтерфейсом. Інструмент для інтеграції з популярними стрімінговими програмами забезпечить автоматичне відображення подяк під час стріму, що додасть додаткового рівня інтерактивності.

1.2 Аналіз аналогів програмного продукту

Основною метою платформи для подяк стрімерам на Twitch є надання творцям контенту можливості легко і швидко почати монетизувати свою діяльність. Система повинна мати усі базові функції для легкого початку використання платформи і достатню гнучкість налаштувань зовнішнього вигляду. Для визначення необхідних функцій, що потрібно реалізувати, і помилок, що необхідно уникнути, розглянемо декілька аналогічних програмних продуктів, які вже певний час перебувають на ринку.

Для цього було обрано такі сервіси:

- Donatello [9];
- Diaka [10];
- Monobank [11].

Donatello – це українська платформа донатів, яка працює за принципом маркетплейсу. Користувачі платформи — автори контенту та їхні підписники.

На рисунку 1.1 зображено домашню сторінку вебсайту.

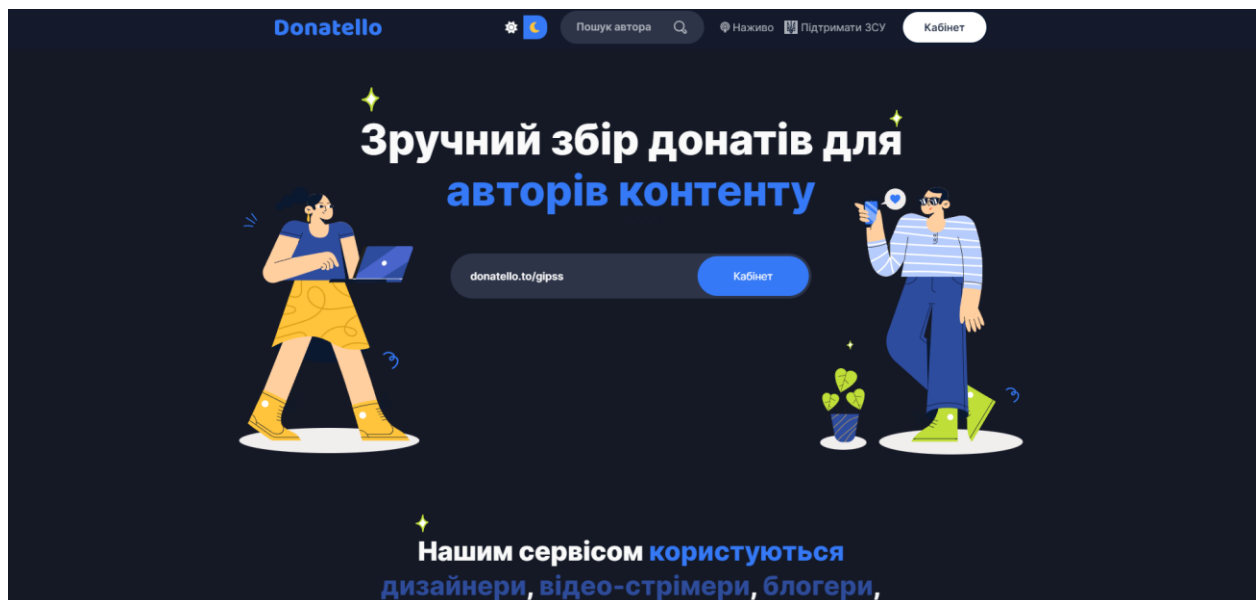


Рисунок 1.1 – Donatello (домашня сторінка)

До переваг цього сервісу належать:

- прогресивний і сучасний дизайн;
- можливість реєстрації за допомогою декількох сторонніх сервісів;
- достатня адаптивність сайту для мобільних пристроїв;
- великий обсяг функцій, який не обмежується лише стрімінговими сервісами.

До недоліків відносяться:

- не дуже зрозумілий інтерфейс і процес налаштування віджетів;
- велика комісія відносно інших сервісів;
- відсутність англійської мови;
- обмеження прибутку для початківців.

Diaka – українська система подяк, яка є доволі простою у використанні і дозволяє новачкам швидко почати монетизувати свою творчість.

На рисунку 1.2 зображено домашню сторінку вебсайту.

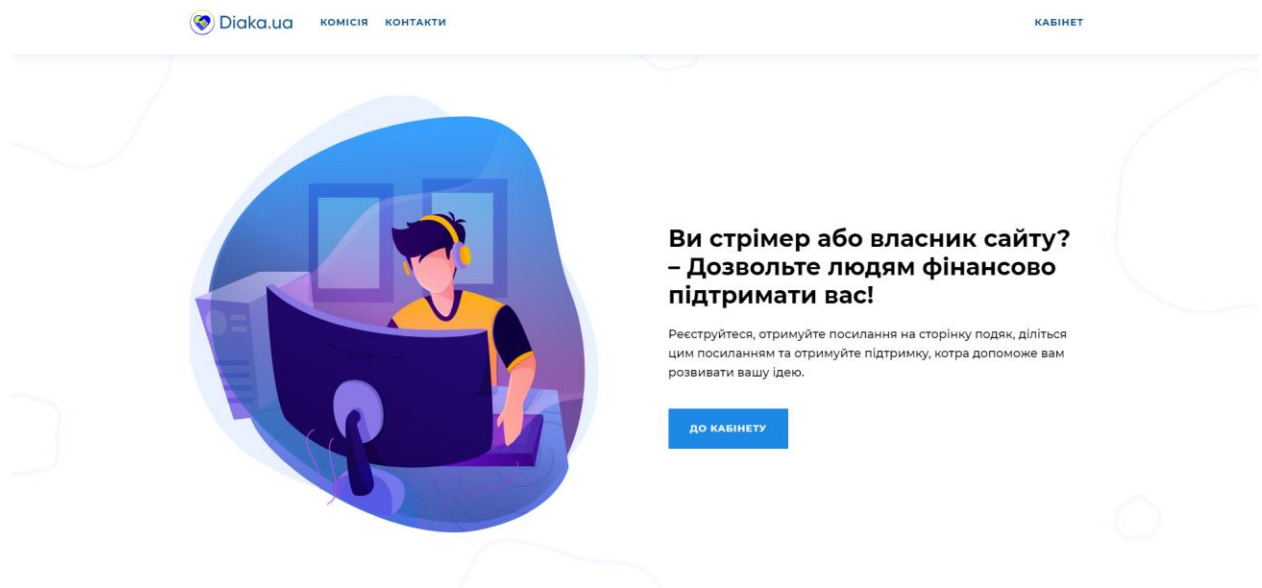


Рисунок 1.2 – Diaka (домашня сторінка)

Перевагами цієї платформи є:

- простий і зрозумілий інтерфейс;
- вхід за допомогою декількох сторонніх сервісів;
- відсутність обмежень прибутку для новачків;
- легкий процес налаштування віджетів;
- чудова адаптивність для мобільних пристроїв.

До недоліків можна віднести:

- сумнівну стабільність сервісу;
- відсутність англійської мови;
- місцями застарілий інтерфейс;
- повільний процес виплати накопичених коштів.

Monobank – це в першу чергу банк, який також надає зручний функціонал для проведення зборів і швидкого отримання коштів на рахунок автора контенту.

На рисунку 1.3 зображено домашню сторінку вебсайту.

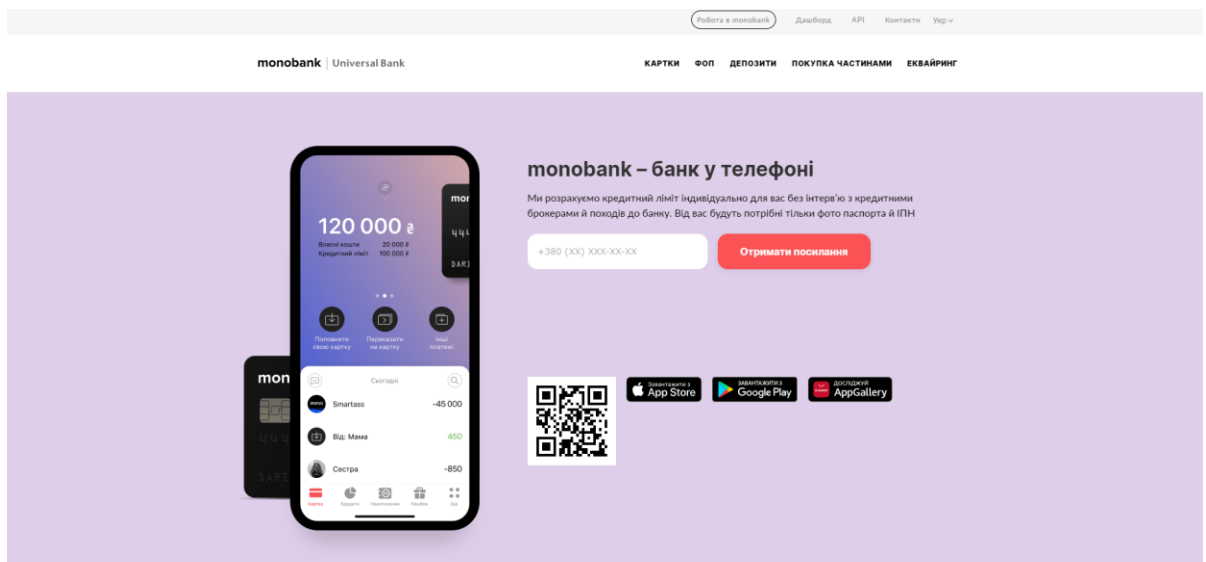


Рисунок 1.3 – Monobank (домашня сторінка)

До переваг цієї платформи можна віднести:

- швидке отримання грошей на рахунок;
- сучасний і зрозумілий інтерфейс;
- багатомовність;
- відсутність комісії.

Недоліками є:

- необхідно бути клієнтом банку;
- низький рівень кастомізації віджетів.

Сформуємо кінцеві дані, отримані з аналізу аналогічних продуктів, у вигляді таблиці.

Таблиця 1.1

Характеристика	Donatello	Diaka	Monobank	Hroshyk
Сучасний дизайн	+	+-	+	+
Зрозумілий дизайн	+-	+	+	+
Адаптивність	+-	+	-	+-
Двомовність	-	-	+	+

Продовження таблиці 1.1

Зручна кастомізація віджетів	+ -	+	-	+
Низька комісія	-	+ -	+	+
Інтеграція з популярними стрімінговими програмами	+	+	+	+
Дохід не обмежено в залежності від часу співпраці з сервісом	-	+	+	+
Стабільність роботи платформи	+	-	+	+ -

Базуючись на цих критеріях, ми отримали такі вимоги до нашої майбутньої платформи:

- сучасний та зрозумілий інтерфейс;
- достатня адаптивність платформи;
- наявність англійської мови;
- зручна і зрозуміла кастомізація віджетів;
- комісія нижча, ніж у конкурентів;
- відсутність обмежень на дохід;
- інтеграція зі стрімінговими програмами;
- стабільна робота платформи.

1.3 Вибір архітектури веборієнтованої платформи для подяк стрімерам на Twitch

Для написання платформи для подяк стрімерам на Twitch було вирішено обрати архітектурний шаблон MVC (Model-View-Controller) [12]. Використання цього шаблону передбачає розподілення логіки на три частини і розміщення основної бізнес-логіки продукту на серверній частині. Це підвищить масштабованість системи та дозволить пришвидшити розробку програмного продукту і зробити її більш зручною.

Вищезгадані три частини архітектурного шаблону MVC (рис. 1.4) – це модель, що представляє собою дані, вид, що відповідає за інтерфейс користувача, і контроллер, який містить бізнес-логіку і поєднує модель та вид між собою.

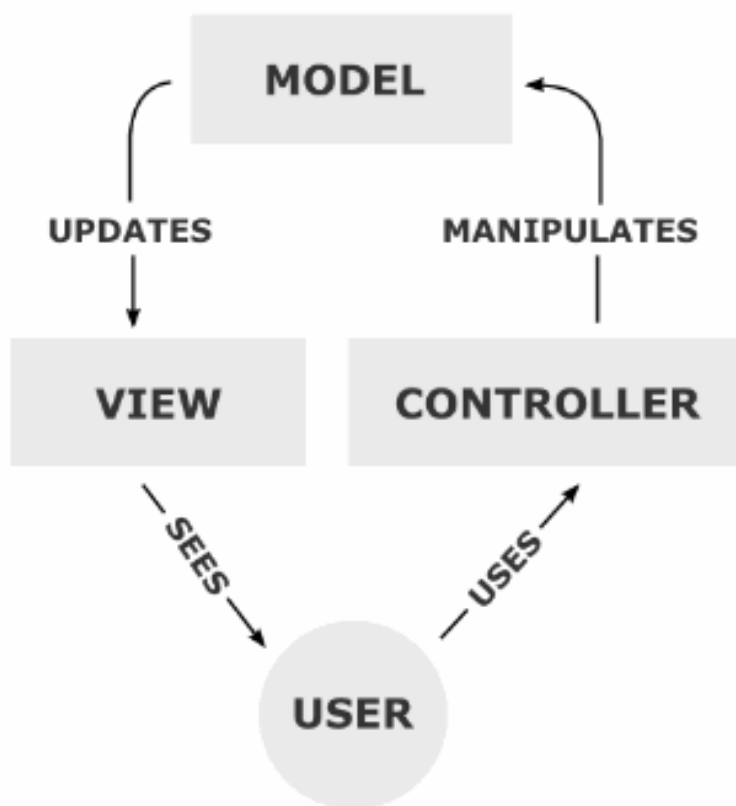


Рисунок 1.4 – Архітектурний шаблон MVC

Для реалізації платформи подяк стрімерам на Twitch також буде використано JavaScript-бібліотеку HTMX [13], оскільки вона допомагає привнести до проекту частину архітектурного шаблону SPA (Single-Page Application) [14]. Це буде зроблено, бо SPA має перевагу в тому, що створює ілюзію безперервності і миттєвості роботи вебсайту.

В результаті, клієнтська сторона буде завжди отримувати спочатку повну статичну HTML-сторінку з підключеною бібліотекою HTMX, яка вже за потреби буде виконувати асинхронні запити на сервер, отримуючи лише невеликі шматочки HTML-коду, що значно підвищить продуктивність вебсайту.

Серверна частина буде реалізована з використанням патерну Стратегія [15], за допомогою якого буде визначатися обсяг даних, що потрібно віддати – повну сторінку чи частину HTML-коду. Це допоможе чітко розділити логіку

роботи для користувача, що тільки зайшов на конкретну сторінку за посиланням, та користувача, який натиснув на кнопку, що надсилає асинхронний запит. Також, серверна частина коду буде відповідати за валідацію і верифікацію даних. Для перевірки чи користувач має доступ до певного ресурсу буде застосовано автентифікацію й авторизацію.

Таким чином, буде написаний програмний продукт, який буде легко масштабувати, підтримувати і його кодова база буде знаходитися в одному місці, що спростить розробку.

1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Для зручного та ефективного написання коду потрібно використовувати сучасні редактори коду і повноцінні середовища розробки, тому для цього було обрано PyCharm [16] – найпопулярніший інструмент розробки на Python.

Для створення серверної частини програмного продукту було обрано мову програмування Python, оскільки вона надає багато можливостей для швидкої реалізації проєктів. Також для серверної частини було обрано фреймворк Django, бо він є найпопулярнішим серед аналогів і має велику кількість додаткових бібліотек.

Для написання клієнтської частини проєкту буде використано стандартний шаблонізатор Django, який опирається на використання HTML, CSS та мови програмування JavaScript. У додачу до цього було обрано UI-фреймворк Bootstrap [17], оскільки він пропонує велику колекцію готових стилів, іконок та компонентів і є найпопулярнішим серед інших схожих інструментів. Також для покращення користувацького досвіду буде використано JavaScript-бібліотеку HTMX, яка забезпечить комфортний досвід користування та пришвидшить роботу клієнтської частини.

Для того, щоб обрати базу даних, яка буде використовуватися для створення платформи, було проведено аналіз найпопулярніших реляційних баз даних і виявлено їхні переваги та недоліки відносно інших.

Було проведено аналіз безпекових механізмів обраних баз даних (табл. 1.2).

Таблиця 1.2

Аналіз безпекових механізмів баз даних

	Автентифікація	Контроль доступу на основі ролей	Шифрування даних	Зашифроване підключення	Захист від brute-force атак
PostgreSQL	+	+	+	+	+
MSSQL	+	+	+	+	-
MySQL	+	+	+	+	-

Також було здійснено пошук інформації щодо максимальної місткості різних сутностей в обраних базах даних (табл. 1.3).

Таблиця 1.3

Максимальна місткість різних сутностей для кожної бази даних

	Рядок	Таблиця	Вся база даних
PostgreSQL	1.6 ТБ	32 ТБ	32 ТБ
MSSQL	8060 Б або ∞	524272 ТБ	524272 ТБ
MySQL	64 КБ	64 ТБ	256 ТБ

Під час детальнішого аналізу обраних СКБД було виявлено, що PostgreSQL надає потужніші можливості для роботи зі складними запитамі та великими обсягами даних завдяки підтримці розширених типів даних та функцій, таких як JSONB для роботи з JSON-даними [18]. Це може бути корисним для зберігання різноманітної інформації про транзакції та користувачів. PostgreSQL має кращу підтримку розширюваності та налаштувань, що дозволяє легко додавати нові функціональні можливості.

Також PostgreSQL забезпечує вищий рівень відповідності стандартам SQL, що полегшує міграцію та інтеграцію з іншими системами. У контексті платформи для подяк стрімерам це дозволяє забезпечити більш надійну та

передбачувану роботу з базою даних, знижуючи ризик помилок при виконанні складних запитів. Крім того, PostgreSQL має покращену підтримку для транзакцій та ACID-властивостей, що критично важливо для забезпечення цілісності даних при обробці фінансових транзакцій.

На відміну від MySQL, PostgreSQL краще справляється з паралельними запитами та має ефективніші механізми блокування, що знижує конкуренцію за ресурси при високих навантаженнях [19]. Це особливо актуально для платформи, яка може обслуговувати велику кількість одночасних користувачів. Також, на відміну від MSSQL, PostgreSQL є повністю безкоштовним і відкритим програмним забезпеченням, що зменшує витрати на ліцензії та забезпечує вільний доступ до вихідного коду для налаштування під специфічні потреби проекту.

Таким чином, вибір PostgreSQL для платформи для подяк стрімерам на Twitch дозволяє забезпечити високу продуктивність, гнучкість, надійність та вигідність у використанні.

Впродовж проектування програмного продукту було визначено мінімальні оптимальні вимоги до серверного обладнання для того, щоб забезпечити надійну і стабільну роботу усієї системи:

- операційна система – Linux;
- процесор з частотою від 2 GHz;
- носій пам'яті об'ємом щонайменше 16 ГБ;
- оперативна пам'ять – 4 ГБ;
- мова програмування Python версії 3.11 та вище.

Висновки до першого розділу

У процесі виконання першого розділу було поставлено задачу на кваліфікаційну роботу, визначено проблеми, які повинен вирішувати програмний продукт, і сформульовано задачі для його створення.

Було проведено дослідження аналогічних систем і платформ, що будуть конкурентами, та виділено їхні основні переваги й недоліки. У результаті цього було сформовано список першочергових цілей нашого програмного продукту: сучасний та зрозумілий інтерфейс, адаптивність платформи, наявність англійської мови, зручна і проста кастомізація віджетів, комісія нижча, ніж у конкурентів, відсутність обмежень на дохід, інтеграція зі стрімінговими програмами і стабільна робота платформи.

Також було обрано й обґрунтовано технології, обрані для реалізації цього проєкту, а саме: Python і фреймворк Django будуть використані для написання серверної частини; PostgreSQL буде базою даних для роботи інформацією стрімерів і донатів; HTML, CSS та JavaScript-бібліотека HTMX – для реалізації клієнтської частини.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ПЛАТФОРМИ ДЛЯ ПОДЯК СТІМЕРАМ НА TWITCH

2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи

Платформа для подяк стрімерам на Twitch повинна вирішувати такі основні проблеми: швидкий і легкий початок використання сервісу, інтуїтивно зрозумілий інтерфейс і відсутність обмежень для доходів. Також програмний продукт повинен надавати можливості для простої та зручної інтеграції зі стрімінговими програмами. Для стрімера обов'язково повинен бути розроблений функціонал виведення коштів на особистий рахунок, але необхідно передбачити введення додаткових даних для подальшого оподаткування доходів платформи і стрімера. Для глядача має бути можливість сплати подяки зручним йому способом, в першу чергу платіжна система зобов'язана підтримувати гривню. Також вебсайт повинен мати базову адаптивність, оскільки багато стрімерів використовують планшети як засоби перегляду донатів.

Типи користувачів:

На платформі наявні три типи користувачів, в яких спосіб використання сервісу повністю відрізняється. Ці типи описано нижче в таблиці 2.1.

Таблиця 2.1

Типи користувачів

Актор	Опис
Стрімер	Цільова аудиторія всього продукту, може переглядати отримані донати, налаштовувати сторінку подяк, може створювати, налаштовувати та інтегрувати віджети для сповіщення про нові донати, може виводити кошти на свій рахунок
Глядач	Може створювати та оплачувати донати, реєстрація не потрібна
Адмін	Має доступ до всієї платформи, в першу чергу зобов'язаний переглядати та опрацьовувати заявки на виведення коштів з гаманця стрімера

Стрімерам на платформі надається ряд можливостей, які вони можуть використовувати. Реєстрація і подальша автентифікація за допомогою акаунт стороннього сервісу Twitch. При реєстрації стрімеру необхідно буде дозволити використання його акаунт для входу, а при наступних відвідуваннях сайту автентифікація буде проводитися автоматично. Також у випадку, якщо стрімер змінив свій нікнейм на стрімінговому сервісі Twitch, то платформа автоматично оновить нікнейм і посилання на сторінку донатів.

Сторінка отриманих донатів, на якій стрімер може переглядати список подяк, шукати і фільтрувати їх за сумою донату, часом відправки та нікнеймом глядача. Також стрімер може натиснути на окремий донат і переглянути його деталі повністю, оскільки повідомлення довжиною більше 20 символів не будуть відображатися повністю у загальному списку.

Можливість редагувати сторінку подяк, змінювати на ній заголовок сторінки, мета-теги, переводити сторінку в тестовий режим, за якого донати не потрібно буде сплачувати, налаштовувати вміст інших властивостей, що стосуються валідацій та плейсхолдерів. Також на цій сторінці стрімер може скопіювати посилання на сторінку подяк, щоб легко використати її в описі свого профілю, та на віджет прогресу збору, щоб глядачі могли бачити свій внесок та користь, яку вони приносять донатами.

На сторінці віджетів подяк стрімер може створити набір конфігурацій, а також налаштувати окремі конфігурації всередині набору. Для цього необхідно вказати максимальну та мінімальну суму донату, на яку буде спрацьовувати це налаштування, а також картинку і звук, які будуть показані на трансляції.

На сторінці платежів стрімер може створити заявку на виведення зароблених коштів. Для цього йому потрібно ввести бажану суму, номер картки та ПІН/РНОКПП. Це зроблено для того, щоб у подальшому ввести систему оподаткування і вести прозору звітність. При першому створенні стрімеру необхідно буде ввести всі ці дані самостійно, але при наступних заявках дані будуть автоматично заповнюватися з останньої транзакції.

Глядач має можливість надсилати подяки стрімерам через їхні сторінки донатів. Для цього йому необхідно ввести його нікнейм, суму, яку він хоче надіслати, та повідомлення, яке відобразиться на екрані. При введенні даних буде застосовуватись валідація, яку налаштував стрімер. Після цього глядача буде переадресовано на сторінку платіжної системи, де йому необхідно ввести дані своєї картки. У разі успішної оплати платіжна система переадресує користувача назад на сторінку подяк стрімера. В іншому випадку подяка залишиться неоплаченою, не відобразиться на трансляції і буде зберігатися в базі даних, доки глядач не сплатить її.

Адміністратор має доступ до всієї платформи, створення, оновлення, видалення будь-яких сутностей, але його основною задачею є перегляд і опрацювання надісланих заявок на виведення коштів. Заявки мають декілька статусів: Створено, В роботі, Виконано, Помилка, Скасовано. В процесі опрацювання заявки адміністратор повинен змінювати цей статус залежно від етапу обробки.

Візуалізація перелічених вище можливостей кожного типу користувача зображена нижче за допомогою діаграми варіантів використання (рис. 2.1).

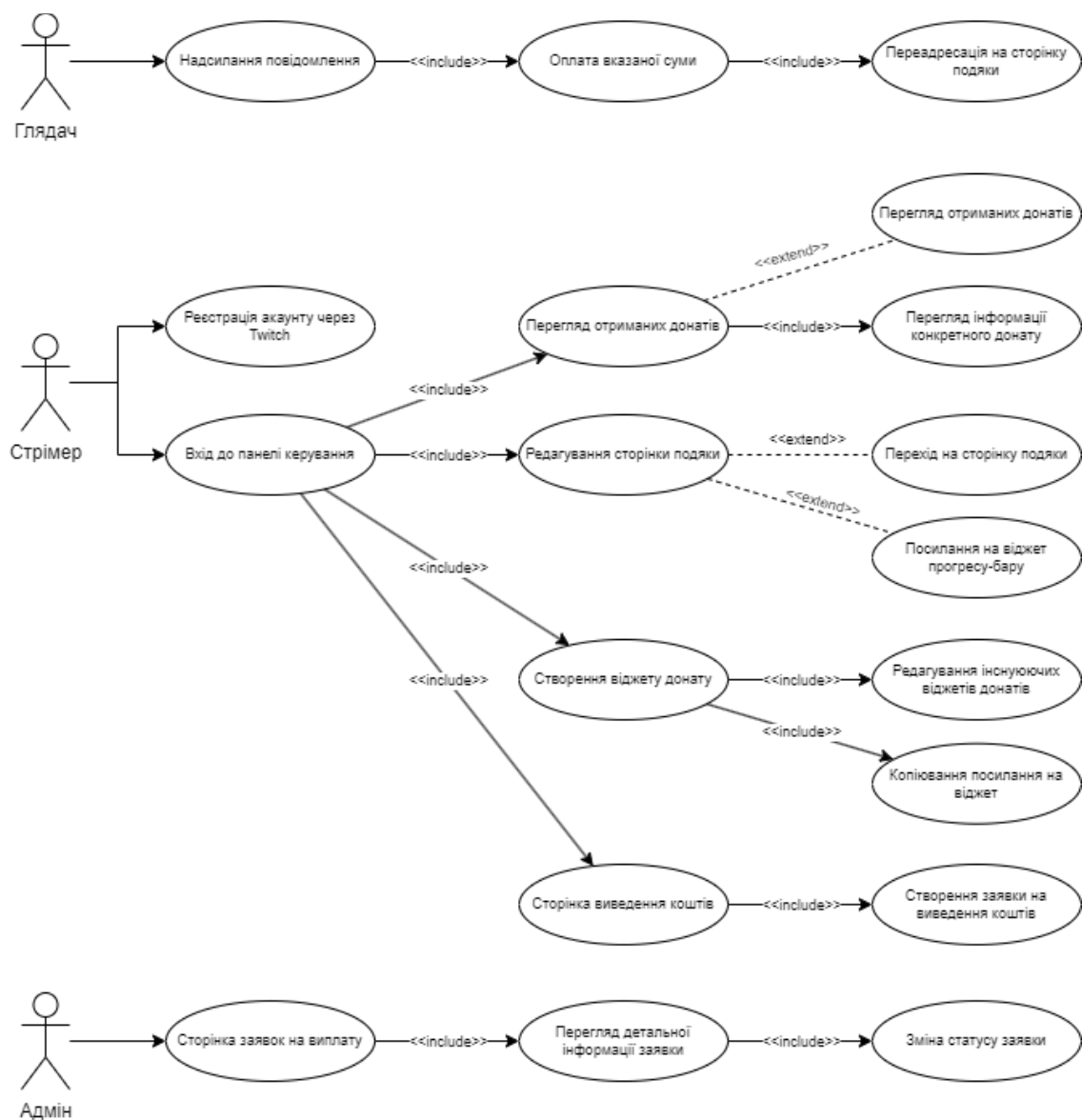


Рисунок 2.1 – Діаграма варіантів використання

Функціональні вимоги:

Платформа для подяк стрімерам на Twitch надає користувачам такі функціональні можливості:

1. Перегляд отриманих донатів, їх фільтрування та детальний огляд.
2. Налаштування вмісту, плейсхолдерів та валідації на сторінці для подяк стрімеру.
3. Копіювання посилань для інтеграції зі стрімінговими програмами.
4. Створення віджетів, їхніх конфігурацій для певних діапазонів значень та оновлення цих даних.
5. Створення заявок на виплату коштів на особистий рахунок.

6. Надсилання та оплата донату з повідомлення від глядача.
7. Огляд надісланих заявок та їхнє опрацювання.

Нефункціональні вимоги:

1. Час навчання для звичайних користувачів – 1-2 години, для досвідчених – 30 хвилин.
2. Швидкість відповіді на прості запити має бути максимум 0.5 секунди та 5 секунд на запити, які є більш складними.
3. Інтуїтивно зрозумілий дизайн.
4. Стабільна робота продукту, яка забезпечить гарантований показ донатів глядачів.
5. Час обслуговування не повинен бути більшим за 5% від загального часу роботи програми.
6. Безперебійна робота протягом 30 робочих днів.

Аналіз функціональних вимог надав нам можливість виділити основні сутності, що зможуть забезпечити реалізацію платформи для подяк стрімерам на Twitch:

- Streamer – сутність, створена для роботи зі стрімера, які зареєструвалися на платформі, їхніми даними та фотографіями із сервісу Twitch;
- DonationPage – сутність, що прив’язана до стрімера видом зв’язку один-до-одного, та є представлення сторінки подяки з динамічним вмістом та валідаціями;
- Donation – сутність, яка представляє собою подяку від глядача стрімеру, що містить нікнейм, суму та повідомлення;
- DonationWidget – це представлення набору конфігурацій для одного віджета;
- Payment – сутність, створена для надсилання стрімерами заявок на виплату накопичених коштів.

Оскільки платформа розроблюється з використанням фреймворку Django і функціоналу, які надають Generic Views на основі класів, то звичайна діаграма класів не підійде для відображення бізнес-логіки платформи. Через те, що у

Django логіка проекту розділяється за допомогою застосунків, різні модулі не мають зв'язків між собою, так як це передбачено архітектурою Django. В такому разі, на рисунку 2.2 буде відображено View-класи та їхні перевизначені методи, в яких логіка за замовчуванням була змінена.



Рисунок 2.2 – Діаграма класів

На діаграмі найбільш важливими є класи ViewerDonationPageView, SuccessfulDonationView та DonationAlertView, тож розглянемо їх детальніше:

- `ViewerDonationPageView` – клас, який відповідає за опрацювання запитів зі сторінки створення донату глядачем.

Метод `get()` в ньому було перевизначено для того, щоб контролювати процес створення форми для сторінки, оскільки модель містить поля, які глядач не повинен вказувати самостійно. Також в цьому методі динамічно додається валідація, налаштована стрімером.

На відправку POST-запиту з цієї сторінки повинна відбуватися логіка інтеграції з платіжною системою, тому метод `post()` було також перевизначено. В ньому потрібно описати логіку генерування посилання для оплати донату і після цього переадресовувати користувача за цим посиланням;

- `SuccessfulDonationView` – це клас, який відповідає за опрацювання успішної оплати певного донату.

Запит на цей маршрут повинен надходити лише зі сторінки платіжної системи. В класі потрібно перевизначити метод `get()` для коректного оновлення балансів і статусів;

- `DonationAlertView` – цей клас відповідає за постійну перевірку нових донатів, які потрібно показати на трансляції.

В ньому буде перевизначено метод `get()` для того, щоб реалізувати логіку знаходження необхідної конфігурації. Оскільки цей маршрут буде дуже часто отримувати запити, необхідно попіклуватися про те, що запити до бази даних є оптимізованими і працюють швидко.

В результаті, така структура задовільнить потреби як стрімера, так і глядача. Вона надасть можливість створювати і редагувати усю необхідну інформацію першому і спростить взаємодію з платформою другому.

2.2 Розробка бази даних системи

Було створено базу даних PostgreSQL, що містить 6 таблиць з усіма необхідними для роботи даними (табл. 2.2).

Таблиці бази даних

Назва таблиці	Призначення
Streamer	Таблиця зареєстрованих стрімерів
DonationPage	Таблиця сторінок подяк, кожна з яких прив'язана до стрімера зв'язком виду один-до-одного
DonationWidget	Таблиця, що представляє собою набір конфігурацій для різних діапазонів
DonationWidgetConfig	Таблиця для конфігурації окремого діапазону
Donation	Таблиця надісланих донатів
Payment	Таблиця створених стрімером заявок на виплату накопичених коштів

Далі наведена діаграма структури бази даних (рис 2.3), назви таблиць на ній відрізняються від вказаних вище, оскільки Django автоматично генерує імена для забезпечення їх унікальності.

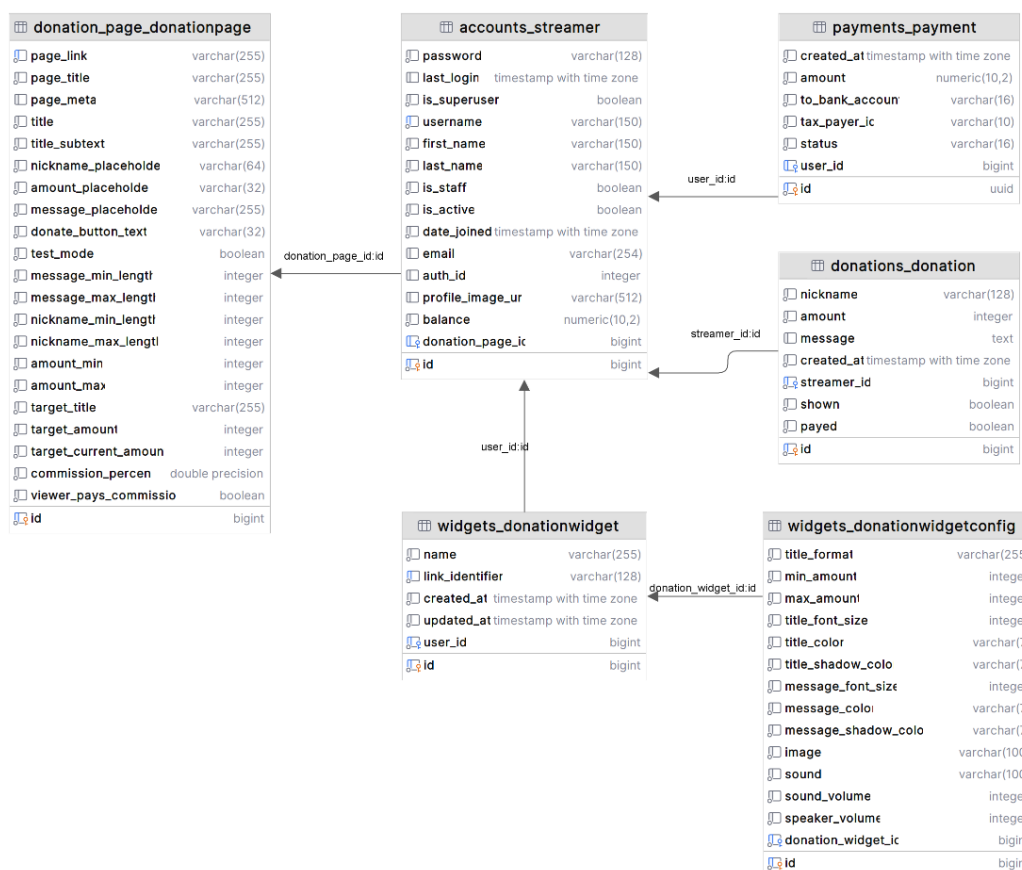


Рисунок 2.3 – ER-діаграма

Далі буде наведено детальніший опис таблиць бази даних.

Таблиця «Streamer» є частково згенерована фреймворком Django, тому не всі з цих полів активно використовуються в проєкті. Ця таблиця передбачена для зберігання інформації про зареєстрованих користувачів. Це можуть бути як стрімери, так і адміністратори. Структура таблиці наведена у таблиці 2.3.

Таблиця 2.3

Таблиця «Streamer»

Назва	Тип даних	PK	FK	Опис поля
id	bigint	+	-	Ідентифікатор
password	varchar(128)	-	-	Пароль
last_login	timestamp	-	-	Дата останнього входу
is_superuser	boolean	-	-	Чи є адміністратором
username	varchar(150)	-	-	Нікнейм
first_name	varchar(150)	-	-	Ім'я користувача
last_name	varchar(150)	-	-	Прізвище користувача
is_staff	boolean	-	-	Чи є робітником платформи
is_active	boolean	-	-	Чи є активним
date_joined	timestamp	-	-	Дата приєднання
email	varchar(254)	-	-	Електронна пошта
auth_id	integer	-	-	Унікальний ідентифікатор акаунту Twitch
profile_image_url	varchar(512)	-	-	Посилання на картинку профіля Twitch
balance	numeric(10,2)	-	-	Баланс користувача
donation_page_id	bigint	-	DonationPage.id	Ідентифікатор сторінки подяк

Таблиця «DonationPage» передбачена для збереження інформації про налаштування сторінки подяк стрімера (табл. 2.4). Вона має зв'язок з таблицею стрімерів виду один-до-одного.

Таблиця 2.4

Таблиця «DonationPage»

Назва	Тип даних	PK	FK	Опис поля
id	bigint	+	-	Ідентифікатор
page_link	varchar(255)	-	-	Посилання на сторінку подяк
page_title	varchar(255)	-	-	Заголовок сторінки подяк
page_meta	varchar(512)	-	-	Мета-теги сторінки подяк
title	varchar(255)	-	-	Заголовок привітання
title_subtext	varchar(255)	-	-	Підзаголовок привітання
nickname_placeholder	varchar(64)	-	-	Плейсхолдер нікнейму

Продовження таблиці 2.4

amount_placeholder	varchar(32)	-	-	Плейсхолдер суми
message_placeholder	varchar(255)	-	-	Плейсхолдер повідомлення
donate_button_text	varchar(32)	-	-	Текст кнопки відправки
test_mode	boolean	-	-	Чи увімкнений тестовий режим
message_min_length	integer	-	-	Мінімальна довжина повідомлення
message_max_length	integer	-	-	Максимальна довжина повідомлення
nickname_min_length	integer	-	-	Мінімальна довжина нікнейму
nickname_max_length	integer	-	-	Максимальна довжина нікнейму
amount_min	integer	-	-	Мінімальна сума подяки
amount_max	integer	-	-	Максимальна сума подяки
target_title	varchar(255)	-	-	Заголовок цілі збору
target_amount	integer	-	-	Сума цілі збору
target_current_amount	integer	-	-	Поточна сума цілі збору
commission_percent	double precision	-	-	Відсоток комісії
viewer_pays_commission	boolean	-	-	Чи глядач платить комісію

Таблиця «DonationWidget» була створена для того, щоб містити інформацію про набір конфігурацій для одного віджета. Саме з цієї таблиці стрімер отримує посилання для інтеграції зі стрімінговими програмами. Структура таблиці наведена нижче (табл. 2.5).

Таблиця 2.5

Таблиця «DonationWidget»

Назва	Тип даних	PK	FK	Опис поля
id	bigint	+	-	Ідентифікатор
name	varchar(255)	-	-	Ім'я набору конфігурацій
link_identifier	varchar(128)	-	-	Унікальне посилання на віджет
created_at	timestamp	-	-	Дата створення
updated_at	timestamp	-	-	Дата останньої зміни
user_id	bigint	-	Streamer.id	Ідентифікатор стрімера, якому належить цей віджет

Таблиця «DonationWidgetConfig» є представлення налаштувань та особливостей для віджета в певному діапазоні суми подяки. Ця таблиця має зв'язок з таблицею DonationWidget виду багато-до-одного. Структура таблиці наведена нижче (табл. 2.6).

Таблиця «DonationWidgetConfig»

Назва	Тип даних	PK	FK	Опис поля
id	bigint	+	-	Ідентифікатор
title_format	varchar(255)	-	-	Формат заголовку
min_amount	integer	-	-	Мінімальна сума подяки
max_amount	integer	-	-	Максимальна сума подяки
title_font_size	integer	-	-	Розмір заголовку
title_color	varchar(7)	-	-	Колір заголовку
title_shadow_color	varchar(7)	-	-	Колір тіні заголовку
message_font_size	integer	-	-	Розмір повідомлення
message_color	varchar(7)	-	-	Колір повідомлення
message_shadow_color	varchar(7)	-	-	Колір тіні повідомлення
image	varchar(100)	-	-	Посилання на зображення
sound	varchar(100)	-	-	Посилання на звук
sound_volume	integer	-	-	Гучність звуку
speaker_volume	integer	-	-	Гучність диктора
donation_widget_id	bigint	-	DonationWidget.id	Ідентифікатор набору конфігурацій

Таблиця «Donation» містить інформацію про надіслані глядачами подяки, інформацію про суму, повідомлення, нікнейм глядача та статуси про те, чи була ця подяка оплачена та показана. Структура таблиці наведена нижче (табл. 2.7).

Таблиця 2.7

Таблиця «Donation»

Назва	Тип даних	PK	FK	Опис поля
id	bigint	+	-	Ідентифікатор
nickname	varchar(128)	-	-	Нікнейм глядача
amount	integer	-	-	Сума подяки
message	text	-	-	Повідомлення подяки
created_at	timestamp	-	-	Дата створення
streamer_id	bigint	-	Streamer.id	Ідентифікатор стрімера, якому подяка була надіслана
shown	boolean	-	-	Чи була подяка показана
payed	boolean	-	-	Чи була подяка сплачена

Таблиця «Payment» містить інформацію про створені стрімерами заявки на виведення коштів, зароблених ними, на особистий рахунок. Ця таблиця містить інформацію про суму виведення та податкові дані стрімера. Структура таблиці наведена нижче (табл. 2.8).

Таблиця 2.8

Таблиця «Payment»

Назва	Тип даних	PK	FK	Опис поля
id	uuid	+	-	Ідентифікатор
created_at	timestamp	-	-	Дата створення
amount	numeric(10,2)	-	-	Сума виведення
to_bank_account	varchar(16)	-	-	Номер картки стрімера
tax_payer_id	varchar(10)	-	-	ІПН/РНОКПП стрімера
status	varchar(16)	-	-	Статус заявки
user_id	bigint	-	Streamer.id	Ідентифікатор стрімера

В результаті, була створена база даних, яка надає усі можливості для коректної роботи продукту і вирішення проблем, поставлених раніше. Всі таблиці займають важливе місце, але найбільш ключовими є DonationWidgetConfig, Donation та Payment, оскільки вони відповідають за зручне та легке у налаштуванні відображення нових подяк на екрані та правильну і надійну роботу з транзакціями і гаранціями користувачів.

2.3 Проектування та реалізація алгоритмів роботи системи

Оскільки платформа була реалізована з використанням шаблону проектування MVC, то проект не має чіткого розподілення між клієнтською та серверною частинами. Майже вся необхідна логіка знаходиться на боці серверу і на кожен запит від користувача за допомогою шаблонів генерується відповідь у вигляді HTML-коду.

Не зважаючи на те, що використовується JavaScript-бібліотека HTMX, логіка програмного продукту залишається незмінною. Клієнтська частина так

само завжди робить запит на сервер, отримуючи відповідь переважно у вигляді HTML-сторінки або шматочку сторінки. Єдина відмінність в тому, що HTMX забезпечує відправлення запит і оновлення сторінки у фоновому режимі, завдяки асинхронним запитам.

Залежно від того, за яким маршрутом переходить користувач, відображаються різні сторінки у відповідь. Таким чином, коли користувач натискає на кнопку авторизації, після цього він бачить панель адміністрації стрімера, а коли натискає на опцію «Платежі» в боковому меню, то бачить створені ним заявки. Детальніший алгоритм роботи платформи візуалізовано на рисунку 2.4.

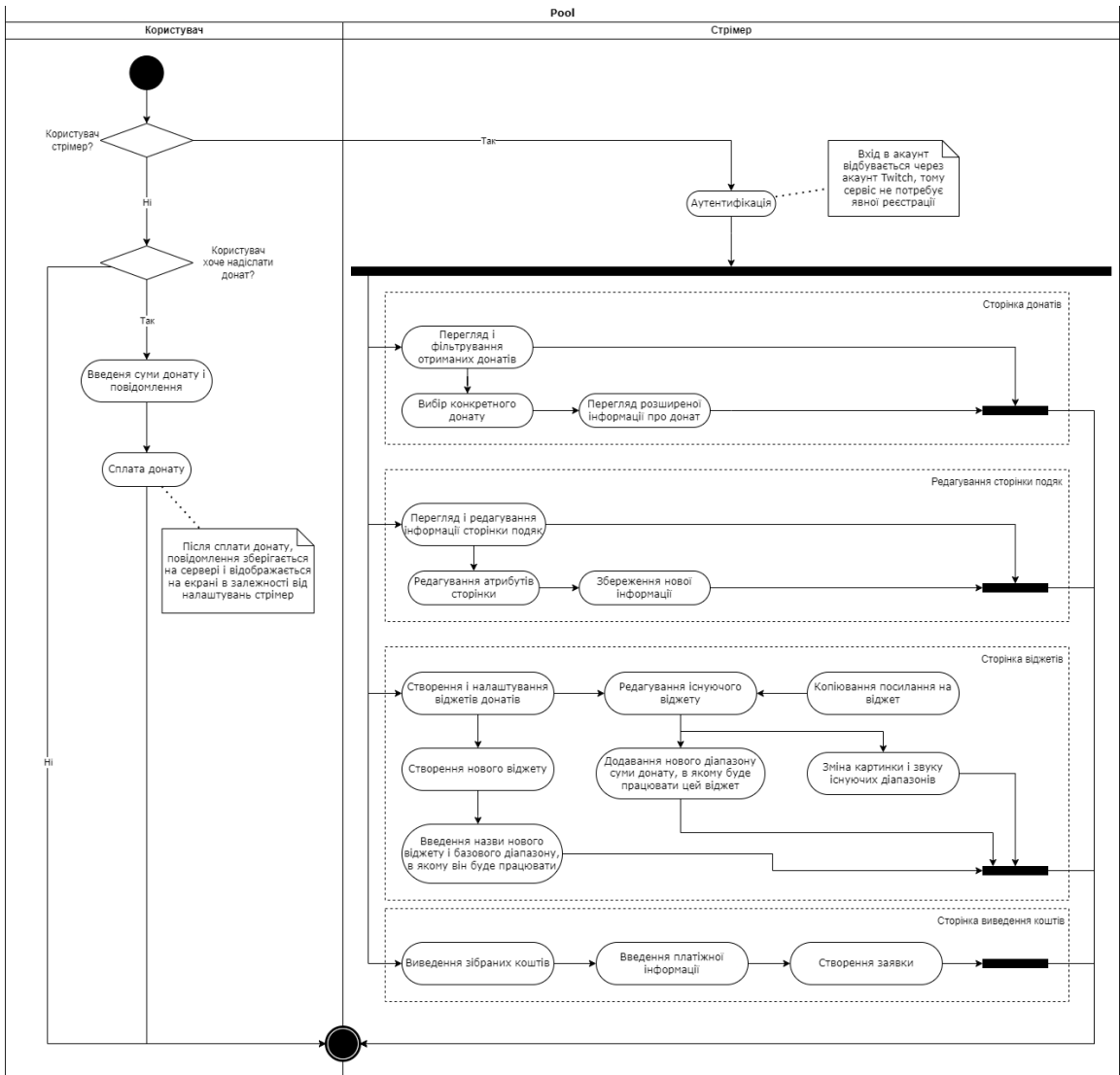


Рисунок 2.4 – Діаграма діяльності

Діаграма на рисунку 2.4 розділена на дві частини, які відповідають за візуалізацію алгоритмів роботи програми для глядача й стрімера.

Для глядача основною функцією є можливість надіслати подяку стрімеру, ввівши свій нікнейм, суму подяки та повідомлення. Після цього глядач переадресовується на сторінку оплати донату. У випадку успішної оплати, вебсайт платіжної системи переадресує глядача назад на сторінку подяки стрімеру.

Для стрімера існує значно більш обширний функціонал, який в першу чергу забезпечує його усіма необхідними інструментами для налаштування сторінки подяки, віджетів, а також перегляду отриманих донатів і доступ до інтеграції зі стрімінговими програмами:

- автентифікація – на платформі процес залежить від стороннього сервісу Twitch, за допомогою якого стрімеру необхідно реєструватися та входити до свого акаунту;
- сторінка отриманих донатів – сторінка, на якій можна переглянути історію подяк, шукати донати за певним нікнеймом, сумою та проміжком часу;
- налаштування сторінки подяки – функціонал, що забезпечує стрімера можливістю встановлювати різноманітні обмеження для нікнейму, суми та повідомлення користувачів;
- сторінка віджетів – сторінка, що надає можливість створювати набори конфігурацій, оновлювати їх, вказувати налаштування для конкретних конфігурацій та отримати посилання для інтеграції зі стрімінговими програмами такими, як OBS;
- сторінка платежів – сторінка, на якій можна створити заявку на виведення накопичених коштів та переглянути статус раніше створених заявок.

2.4 Реалізація платформи для подяк стрімерам на Twitch

Операція, що відбувається в першу чергу при відвідуванні сайту користувачем, це перевірка на те, чи є він в нашій системі – автентифікація. Тому спочатку розглянемо саме цей функціонал (рис. 2.5).

```
class TwitchBackend(BaseTwitchBackend):
    Explain Code Generate Tests Generate Docstrings Ask Sourcery GIPSS
    def authenticate(self, request, token=None):
        response = requests.get(
            TWITCH_USERS_URL,
            headers={
                "Accept": "application/vnd.twitchtv.v5+json",
                "Client-ID": TWITCH_CLIENT_ID,
                "Authorization": f"Bearer {token}",
            },
        )
        if response.ok:
            user_information = response.json()["data"][0]
            user_data = {
                TWITCH_USERNAME_FIELD: user_information["login"],
                TWITCH_EMAIL_FIELD: user_information.get("email"),
                "auth_id": user_information["id"],
                "profile_image_url": user_information.get("profile_image_url"),
            }
            try:
                user = User.objects.get(auth_id=user_data["auth_id"])
                modified = False
                for field_name in (TWITCH_USERNAME_FIELD, TWITCH_EMAIL_FIELD):
                    if getattr(user, field_name) != user_data[field_name]:
                        setattr(user, field_name, user_data[field_name])
                        modified = True
                if modified:
                    user.donation_page.page_link = user_data[TWITCH_USERNAME_FIELD]
                    user.donation_page.save()
                    user.save()
            except User.DoesNotExist:
                user = User.objects.create_user(**user_data)
            return user
        return None
```

Рисунок 2.5 – Функціонал автентифікації

Автентифікація була реалізована з використанням стороннього сервісу Twitch, оскільки це буде найбільш зручним способом реєстрації та входу до акаунту для більшості стрімерів. Оскільки фреймворк Django значну частину автентифікації та авторизації проводить автоматично, мені необхідно було описати лише певні її частини. А саме клас TwitchBackend, який відповідає за

використання токєну, наданого користувачем, для його надсилання на сервер Twitch і перевірки чи токен є правильним і такий користувач існує в нашій базі даних. У випадку, якщо користувач є новим, його буде створено, а якщо користувач вже є зарєєстрованим на нашій платформі, але змінив нікнейм чи фотографію, то вони зміняться автоматично. Наступним кроком буде використання цього класу в налаштуваннях Django (рис. 2.6). Усе інше вже автоматизоване фреймворком, наприклад, збереження інформації про користувача на боці клієнта з використанням сесій та cookies.

```
AUTHENTICATION_BACKENDS = [
    "django.contrib.auth.backends.ModelBackend",
    "accounts.auth.TwitchBackend",
]
```

Рисунок 2.6 – Використання власного класу для автентифікації

Також нижче продемонстровано створення користувача у випадку реєстрація, оскільки це є важливою частиною входу за допомогою акаунту Twitch (рис 2.7).

```
class StreamerManager(BaseUserManager):
    Explain Code  Generate Tests  Generate Docstrings  Ask Sourcery  1 usage (1 dynamic)
    def create_user(self, username, email=None, password=None, **extra_fields):
        if not username:
            raise ValueError("Users must have a username")
        if not password:
            password = self.make_random_password()
        password = hash_password(password)
        new_user = self.model(
            username=username,
            email=email,
            password=password,
            is_staff=False,
            is_superuser=False,
            **extra_fields,
        )
        new_user.save()
        donation_page = DonationPage.objects.create(page_link=username)
        donation_page.save()
        new_user.donation_page = donation_page
        new_user.save()
        return new_user
```

Рисунок 2.7 – Перевизначення методу створення користувача

За замовчуванням в Django вже продумано дуже схожу логіку і в деяких випадках цей метод не потрібно перевизначати. Але оскільки згідно зі спроектованими вимогами кожен стрімер повинен мати зв'язану з ним сторінку подяки, цей метод було доповнено створенням відповідної сторінки.

Одною з найбільш важливих ланок проекту є надсилання донату глядачем. За це в коді відповідає клас `ViewerDonationPageView`. Коли глядач тільки заходить на цю сторінку, клієнт надсилає запит з методом GET. У відповідь йому відображається форма для заповнення даних. Відповідний код зображено нижче на рисунку 2.8.

```
class ViewerDonationPageView(CreateView):
    template_name = "donation_page/viewer_donation_page.html"
    form_class = DonationForm
    object: Donation | None

    def get(self, request, *args, **kwargs):
        link = kwargs["link"]
        response = super().get(request, *args: *args, **kwargs)
        response.context_data["donation_page"] = get_object_or_404(DonationPage, page_link=link)
        form = response.context_data["form"]
        obj = response.context_data["donation_page"]
        donation_add_validation(form, obj)
        return response
```

Рисунок 2.8 – Отримання сторінки з формою надсилання подяки

Даний метод робить пошук в базі даних на відповідну сторінку подяки. У випадку, якщо сторінку подяки не було знайдено, сервер відповість помилкою з кодом статусу 404.

Також важливою частиною цього є функція `donation_add_validation` (рис. 2.9), оскільки саме в ній відбувається додавання валідацій, вказаних стрімером. В першу чергу від цього залежить, яку мінімальну та максимальну суму може надсилати глядач.

```
def donation_add_validation(form, obj):
    for field_name in ("nickname", "amount", "message"):
        field: Field = form.fields[field_name]
        field.widget.attrs["placeholder"] = getattr(obj, f"{field_name}_placeholder")
        if isinstance(field, CharField):
            field.widget.attrs["maxlength"] = getattr(obj, f"{field_name}_max_length")
            field.widget.attrs["minlength"] = getattr(obj, f"{field_name}_min_length")
        elif isinstance(field, IntegerField):
            field.widget.attrs["max"] = getattr(obj, f"{field_name}_max")
            field.widget.attrs["min"] = getattr(obj, f"{field_name}_min")
```

Рисунок 2.9 – Функція для динамічного додавання валідації на форму надсилання донату

Також частиною класу `ViewerDonationPageView` є метод `post`, який спрацьовує у випадку отримання запиту з методом `POST`. В середині цього методу реалізована основна частина інтеграції з платіжною системою `LiqPay` [20]. Хоч ця платіжна система і має Python-бібліотеку для легкої інтеграції, публічні методи, представлені в ній, не задовольняли потреби платформи. Тому було прийнято рішення вивчити інші методи бібліотеки для напівручного створення необхідного посилання. В результаті було написано метод, який зображено на рисунку 2.10. Після завершення його роботи глядач переадресовується на сторінку платіжної системи, де він може провести оплату подяки.

```
def post(self, request, *args, **kwargs):
    self.object = None
    form = FullDonationForm(**self.get_form_kwargs())
    result = self.form_valid(form) if form.is_valid() else self.form_invalid(form)
    liqpay = LiqPay(settings.LIQPAY_PUBLIC_KEY, settings.LIQPAY_PRIVATE_KEY)
    request_data = liqpay._prepare_params(
        {
            "action": "pay",
            "amount": str(self.object.amount),
            "currency": "UAH",
            "description": "Donation to a streamer",
            "order_id": str(self.object.pk),
            "version": "3",
            "result_url": request.build_absolute_uri(
                reverse(
                    viewname="successful_donation",
                    kwargs={"pk": self.object.pk, "link": self.object.streamer.donation_page.page_link},
                )
            ),
        }
    )
    params = {"data": liqpay.data_to_sign(request_data)}
    params["signature"] = liqpay._make_signature(*args: liqpay._private_key, params["data"], liqpay._private_key)
    url = urljoin(liqpay._host, url: "3/checkout/") + f"?data={params['data']}&signature={params['signature']}"
    return redirect(url)
```

Рисунок 2.10 – Інтеграція з платіжною системою `LiqPay`

У разі успішної сплати донату, глядач переадресовується зі сторінки платіжної системи назад на нашу платформу. За обробку цього відповідає клас `SuccessfulDonationView`, який оновлює статус створеного раніше донату, а також баланси стрімера і цілі, на яку він збирає кошти (рис. 2.11).

```
class SuccessfulDonationView(SingleObjectMixin, View):
    model = Donation

    def get(self, request, *args, **kwargs):
        self.object = self.get_object()
        self.object.payed = True
        self.object.streamer.balance += self.object.amount
        self.object.streamer.donation_page.target_current_amount += self.object.amount
        self.object.save()
        self.object.streamer.save()
        self.object.streamer.donation_page.save()
        return redirect(reverse(viewname="viewer_donation_page", kwargs={"link": kwargs.get("link")}))
```

Рисунок 2.11 – Опрацювання успішної оплати донату

Цього модуля також стосується JavaScript-код, робота якого направлена в першу чергу на зручність використання платформи для стрімера. Оскільки в панелі адміністрування стрімер повинен мати можливість попереднього огляду сторінки подяки, туди було додано блок з використанням тегу `iframe`. Таким чином, це повністю імітує реальну сторінку, яку у подальшому буде бачити глядач. Але недоліком цього стало те, що при натисканні кнопки створення стрімера перенаправляло на іншу сторінку, і це створювало негативний досвід користування платформою.

Для вирішення цієї проблеми було написано код, що перевіряє чи використовується сторінка подяки як `iframe`. Якщо так, то JavaScript-код заміняв справжню кнопку створення на її звичайну копію, яка при натисканні нічого не робить. Код цього функціоналу зображено на рисунку 2.12.

```
window.onload = () : void => {
    const isIframe : boolean = window.location !== window.parent.location
    const containerBtn : Element = document.querySelector( selectors: "#main-wrapper > div > div.donation-page-form-container > form > div.submit-btn")
    const submitBtn : HTMLButtonElement = containerBtn.querySelector( selectors: "button")
    if (isIframe) {
        const twitchBtn : HTMLDivElement = document.createElement( tagName: "div")
        twitchBtn.innerText = gettext('Send Hroshyk') // gettext is a function from Django
        twitchBtn.classList.add("twitch-btn")
        submitBtn.remove()
        containerBtn.appendChild(twitchBtn)
    }
}
```

Рисунок 2.12 – Перевірка на використання сторінки подяки

Наступної після інтеграції з платіжною системою по важливості є інтеграція зі стрімінговими програмами. За це у коді відповідає клас DonationAlertView (рис. 2.13). Після першого запиту за цим маршрутом, який зазвичай робить стрімінгова програма, метод get повертає повністю порожню сторінку лише з невеликою кількістю JavaScript-коду (рис. 2.14). Одразу після цього сторінка автоматично починає робити запити на сервер з інтервалом 2 секунди для того, щоб перевірити чи не з'явився новий донат, який потрібно показати на екрані. Таким чином, було реалізовано процесу «опитування» (polling), який є альтернативою для використання вебсокетів (websockets).

```
class DonationAlertView(DetailView):
    model = DonationWidget
    template_name = "widgets/donation_widget/donation_alert.html"

    def get(self, request, *args, **kwargs):
        if not request.htmx:
            return super().get(request, *args, **kwargs)

        self.object = self.get_object()
        ranges = {config.pk: (config.min_amount, config.max_amount) for config in self.object.configs.all()}

        donation_to_show = (
            Donation.objects.filter(streamer_id=self.object.user_id, shown=False, payed=True)
            .filter(reduce(lambda x, y: x | y, [Q(amount__range=range_) for range_ in ranges.values()]))
            .order_by("created_at")
            .first()
        )

        if not donation_to_show:
            return HttpResponse("")

        config_id = next(
            (id_ for id_, range_ in ranges.items() if range_[0] <= donation_to_show.amount <= range_[1]),
            None,
        )
        config = next((config for config in self.object.configs.all() if config.pk == config_id), None)

        if not config:
            return HttpResponse("No config found for this donation amount")

        donation_to_show.shown = True
        donation_to_show.save()

        template_name = "widgets/donation_widget/donation_alert_content.html"

        return render(request, template_name, context={"donation": donation_to_show, "config": config})
```

Рисунок 2.13 – Постійна перевірка на існування нових донатів

```

<div id="alert-container"
  hx-get="{% url 'donation_alert' object.link_identifier %}"
  hx-trigger="poll-alert"
  hx-swap="innerHTML transition:true"
>
</div>
<script>
  var timeoutId = null;
  var triggerTimeout = () => {
    timeoutId = setTimeout(function() {
      htmx.trigger('#alert-container', 'poll-alert')
      timeoutId = setTimeout(triggerTimeout, 2000)
    }, 2000)
  }
  window.onload = () => {
    triggerTimeout()
  }
</script>

```

Рисунок 2.14 – Автоматичний початок процесу «опитування» серверу

Після того, як сторінка почала процес «опитування», сервер це відслідковує і виконує інші дії. Тепер на кожен запит він перевіряє чи з'явилася нова подяка і, якщо так, то відповідає видозміненим вмістом з інформацією, яку надіслав користувач та налаштував стрімер. В момент, коли видозмінена сторінка завантажилася, одразу починає звучати медіа, яке обрав стрімер. Після цього JavaScript-код поновлює «опитування» серверу і цикл продовжується як раніше. Код функціоналу зображено на рисунку 2.15.

```

{% load widget_filters %}

<div class="donation-alert-container">
  
  <h1 style="...">
  >
    {% if donation %}
      {% format_string config.title_format nickname=donation.nickname amount=donation.amount %}
    {% else %}
      {% format_string config.title_format nickname="Anonymous" amount=100 %}
    {% endif %}

  </h1>
  <div style="...">
  >
    {% if donation %}
      {{ donation.message|safe }}
    {% else %}
      Текст повідомлення
    {% endif %}
  </div>

  {% if donation %}
    <audio id="audio" src="{{ config.sound.url }}"></audio>
    <script>
      clearTimeout(timeoutId)
      var audio = document.getElementById("audio")
      audio.volume = {{ config.sound_volume }} / 100
      audio.play()
      audio.onended = function() {
        triggerTimeout()
        document.getElementById('alert-container').innerHTML = ''
      }
    </script>
  {% endif %}
</div>

```

Рисунок 2.15 – Шаблон HTML-коду для нового отриманого донату

Також корисним функціоналом, що було розроблено для платформи це двомовність [21]. Вона була реалізована за допомогою інструментів Django. Для цього необхідно було створити словник з перекладам відповідних слів і використати в шаблонах Django ключі, за якими можна знайти переклад. Реалізацію продемонстровано на рисунку 2.16 та 2.17.

```

#: accounts/models.py:12
msgid "Email address"
msgstr "Електронна адреса"

#: accounts/models.py:14
msgid "Profile image url"
msgstr "Посилання на зображення профілю"

#: accounts/models.py:15
msgid "Balance"
msgstr "Баланс"

#: accounts/models.py:18 templates/nav/sidebar.html:18
msgid "Donation page"
msgstr "Сторінка подяк"

#: donation_page/forms.py:61 donation_page/models.py:9
msgid "Page link"
msgstr "Посилання на сторінку"

```

Рисунок 2.16 – Словник перекладів

```

{% load i18n %}

<div class="donation-details">
  <table class="donations-table">
    <tr>
      <th style="...">{% trans 'Nickname' %}</th>
      <td>{{ donation.nickname }}</td>
    </tr>
    <tr>
      <th>{% trans 'Amount' %}</th>
      <td>{{ donation.amount }}</td>
    </tr>
    <tr>
      <th>{% trans 'Date' %}</th>
      <td>{{ donation.created_at|date:'d.m.Y H:i' }}</td>
    </tr>
    <tr>
      <th>{% trans 'Message' %}</th>
      <td>{{ donation.message }}</td>
    </tr>
  </table>
</div>

```

Рисунок 2.17 – Приклад використання словника перекладів

Висновки до розділу 2

В ході виконання другого розділу кваліфікаційної роботи було проаналізовано різні варіанти використання платформи та її функцій. На основі цього було побудовано діаграми діяльності користувачів таких, як Глядач, Стрімер та Адміністратор. Також було визначено функціональні та нефункціональні вимоги, що допомогло спланувати майбутню структуру коду та модулів.

У цьому розділі також було спроектовано структуру бази даних з усіма таблицями, що забезпечать зберігання необхідного обсягу інформації та задовільняють потреби користувача платформи. База даних містить 6 таблиць, які є представленнями зареєстрованих стрімерів, адміністраторів, надісланих подяк, конфігурацій для сторінок та віджетів, а також заявок на виплату коштів, накопичених стрімером.

В останньому пункті було детально розглянуто код програмного продукту, який в основному зберігається на серверній частині платформи. Було вивчено блоки коду, що відповідають за інтеграцію з платіжною системою та

стрімговими програмами. Також було знайдено та обґрунтовано рішення проблем, які виникали впродовж написання коду.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ПЛАТФОРМОЮ ДЛЯ ПОДЯК СТІМЕРАМ НА TWITCH

3.1 Порядок встановлення та налаштування параметрів платформи

Діаграма розгортання – це візуалізація вузлів програмного коду, їх компонентів та об'єктів, що на них обчислюються. Це один із типів структурних діаграм, що використовуються для моделювання фізичних аспектів об'єктно-орієнтованої системи. Діаграма розгортання дозволяє наочно показати, як компоненти системи розташовані на обчислювальних вузлах і взаємодіють між собою.

Діаграма розгортання з візуалізацією вузлів програмного коду зображено на рисунку 3.1.

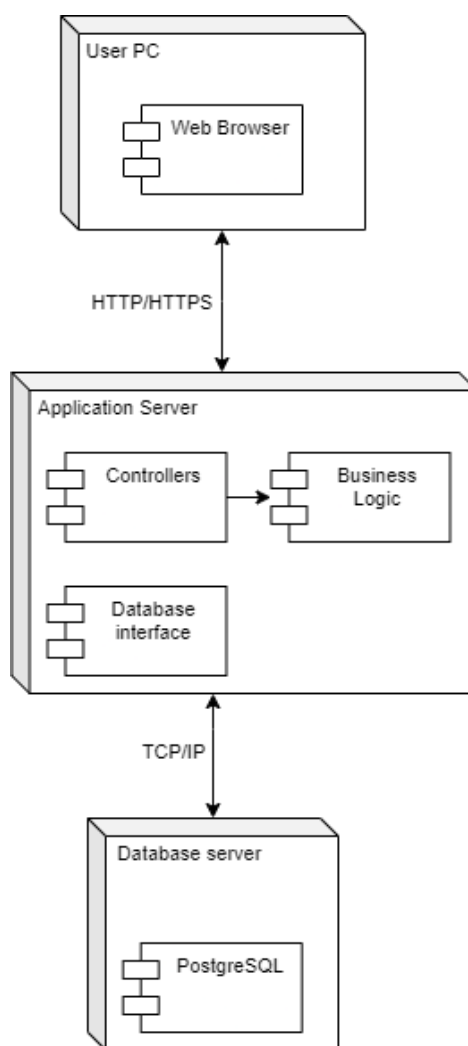


Рисунок 3.1 – Діаграма розгортання

Опис процесу розгортання:

1. База даних PostgreSQL, що за замовчуванням розгортається на порті 5432, до якої буде підключена серверна частина проекту для збереження даних стрімерів та подяк.
2. Серверна частина проекту, що за умов локального розгортання запускається на порті 8000, а за умов віддаленого розгортання – на порті 433. На ньому буде вся бізнес-логіка проекту, створення акаунтів, подяки, виведення коштів та інші CRUD-операції.

3.2 Структура інтерфейсу платформи для подяк стрімерам на Twitch

У випадку, якщо користувач вперше заходить на платформу і він ще є не зареєстрованим, то його зустрічає сторінка для гостей, що запрошує зареєструватися і почати використання платформи (рис. 3.2).

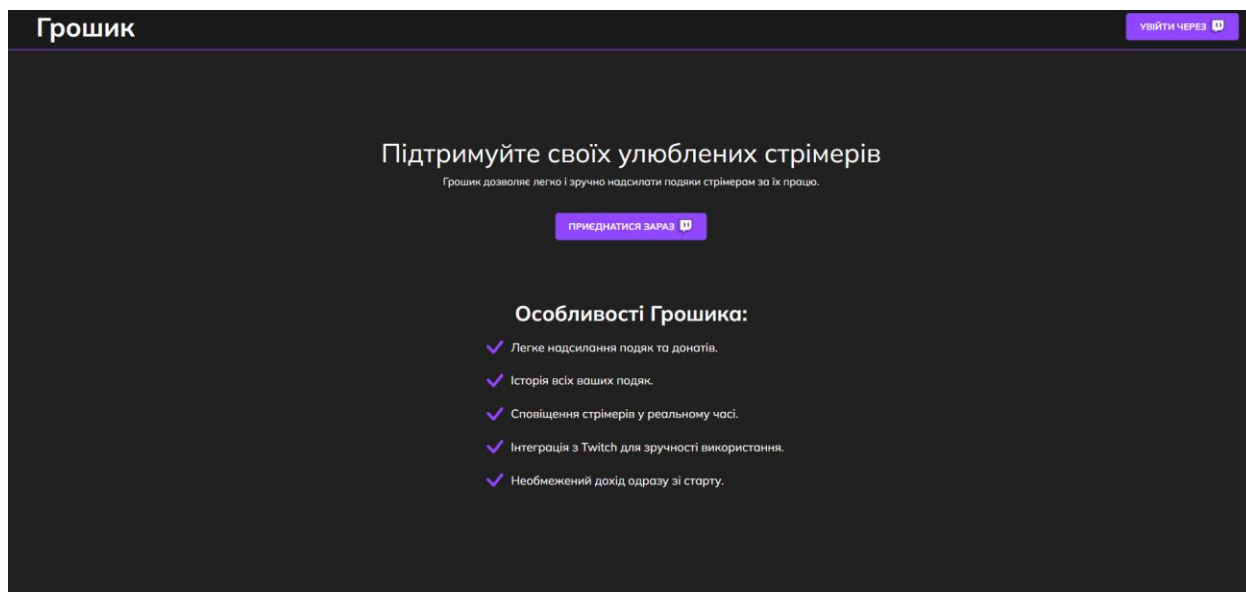


Рисунок 3.2 – Домашня сторінка платформи

З цієї сторінки користувач має можливість зареєструватися на платформі за допомогою акаунту Twitch. При натисканні однієї з запрошувальних кнопок користувача буде переадресовано на сторінки автентифікації Twitch. У випадку, якщо це повністю новий користувач, то йому необхідно буде також надати доступ Twitch-застосунку Грошик для отримання його даних.

Після цього домашня сторінка трохи зміниться свій вигляд, замінив кнопку в заголовку на меню з опціями, а кнопку по центру – на перехід до панелі управління (рис. 3.3).

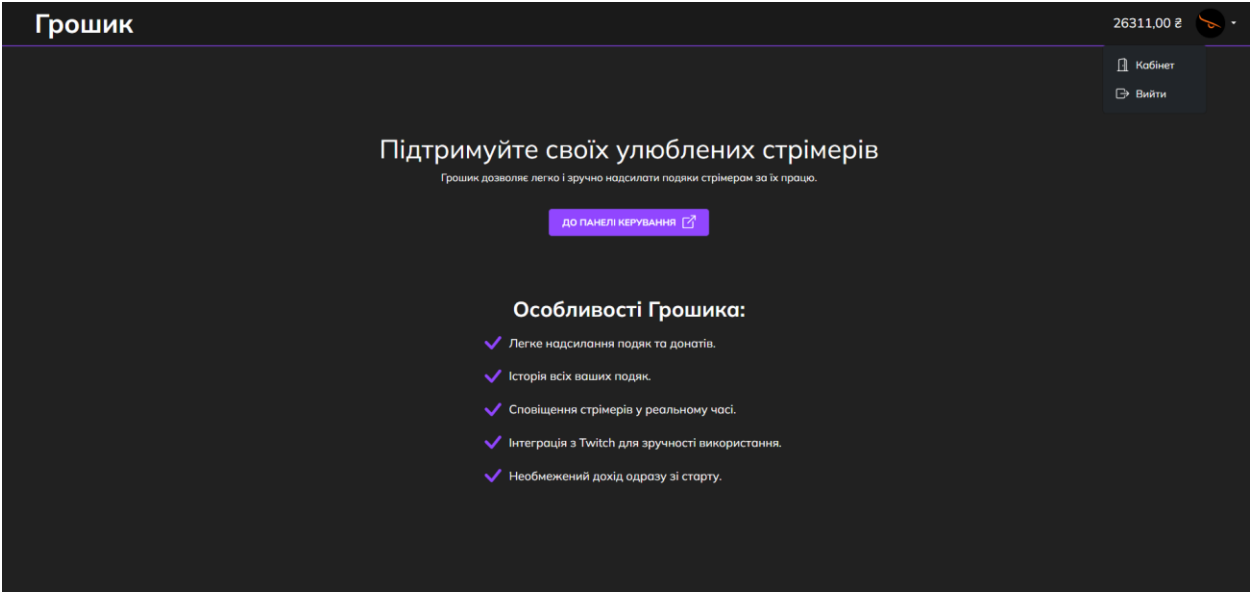


Рисунок 3.3 – Домашня сторінка платформи для автентифікованого користувача

Після входу до свого кабінету стрімер опиняється на сторінці останніх отриманих донатів (рис. 3.4). Тут він може фільтрувати подарунок за нікнеймом глядача, що надіслав їх, сумою та датою і часом надсилання донату. Ці можливості продемонстровані на рисунку 3.5, рисунку 3.6, рисунку 3.7 та рисунку 3.8.

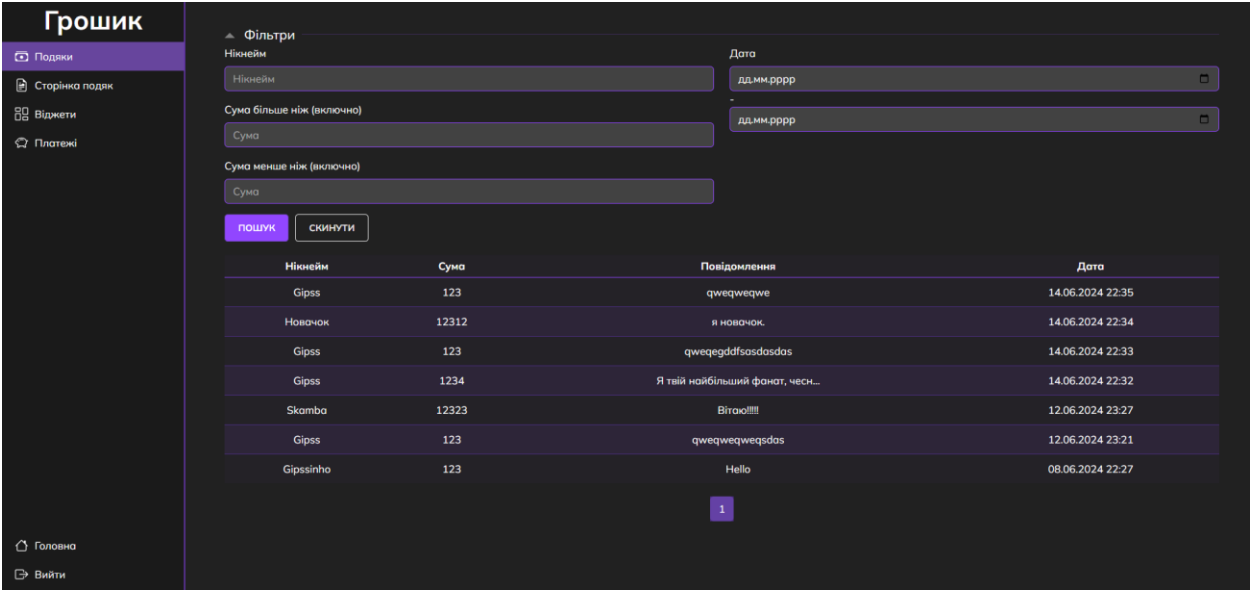


Рисунок 3.4 – Сторінка отриманих донатів без застосованих фільтрацій

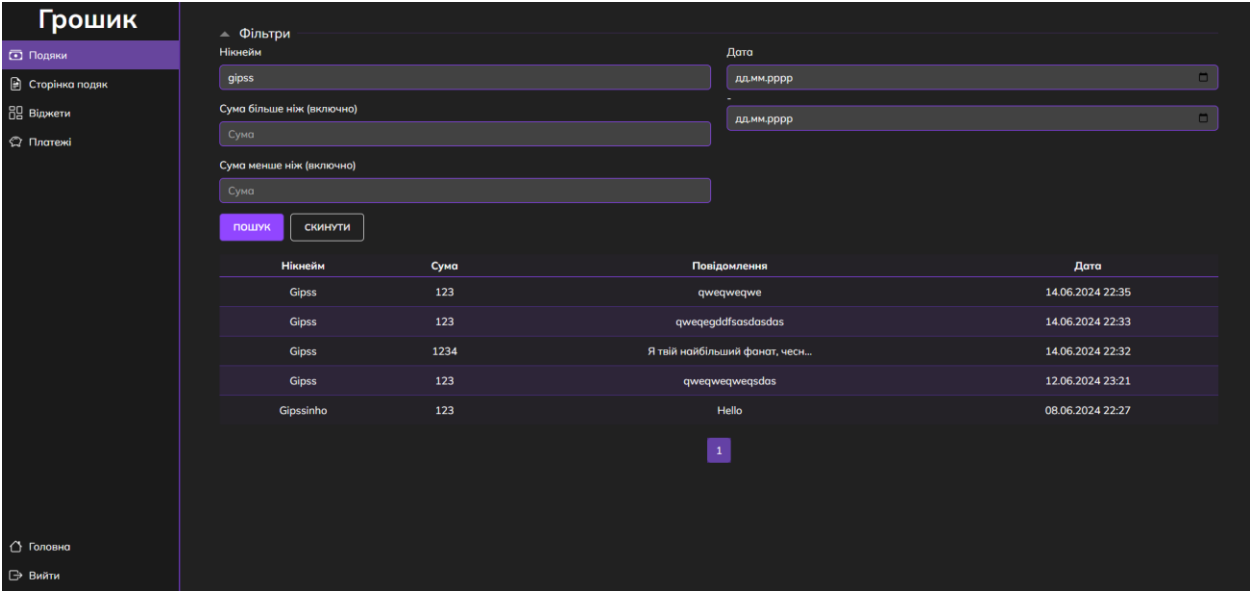


Рисунок 3.5 – Сторінка отриманих донатів із застосованою фільтрацією по нікнейму

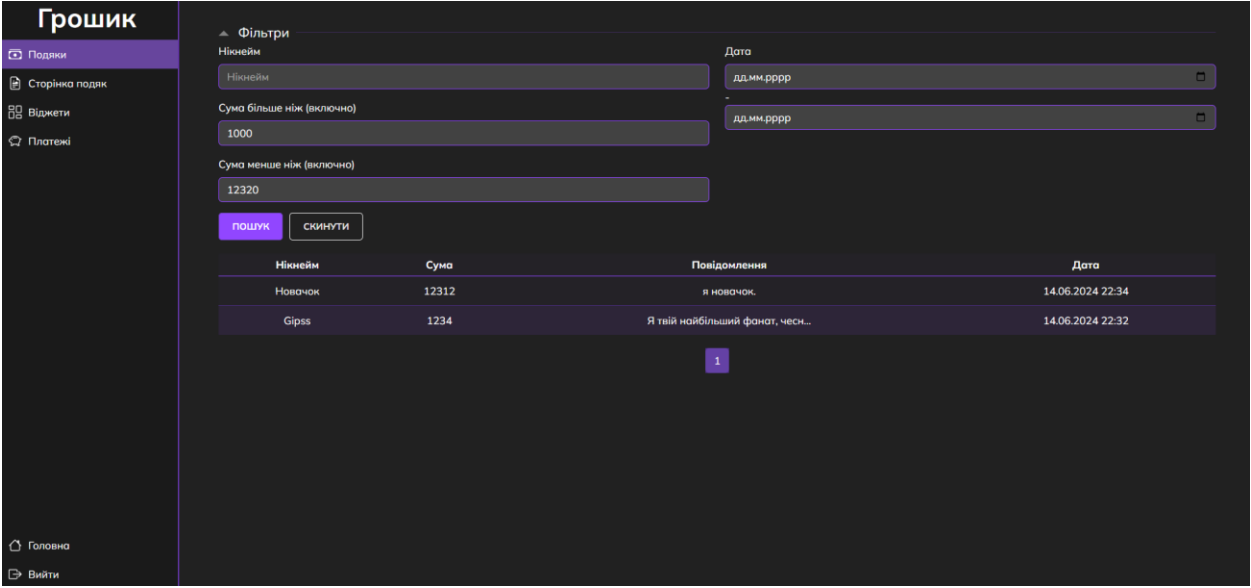


Рисунок 3.6 – Сторінка отриманих донатів із застосованою фільтрацією по сумі

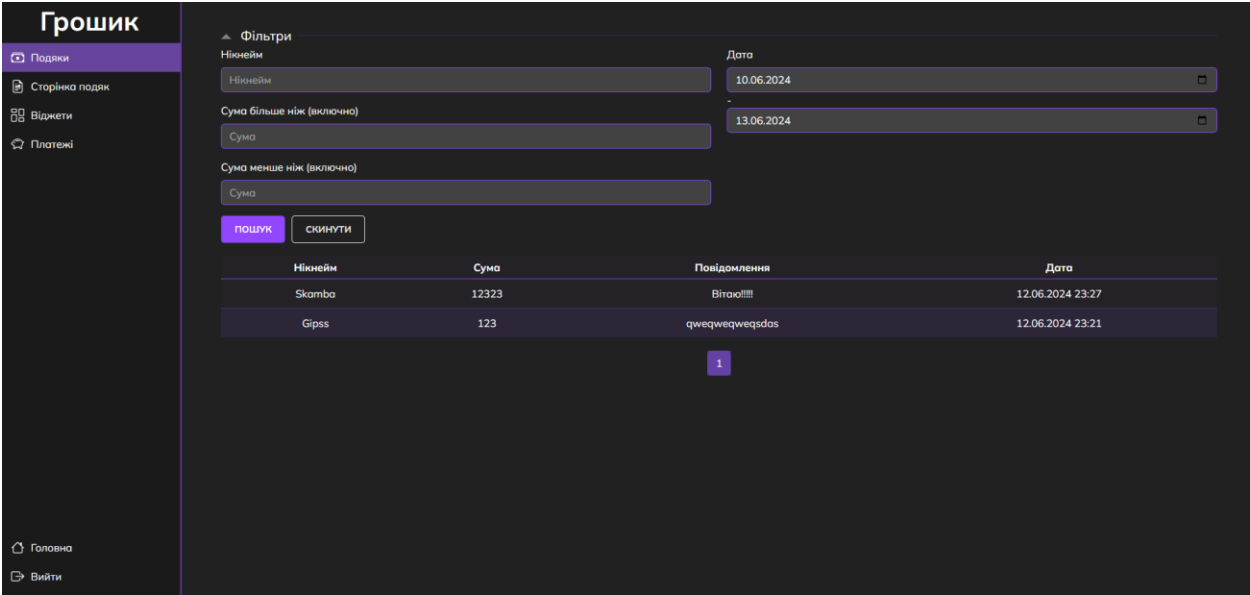


Рисунок 3.7 – Сторінка отриманих донатів із застосованою фільтрацією по даті створення

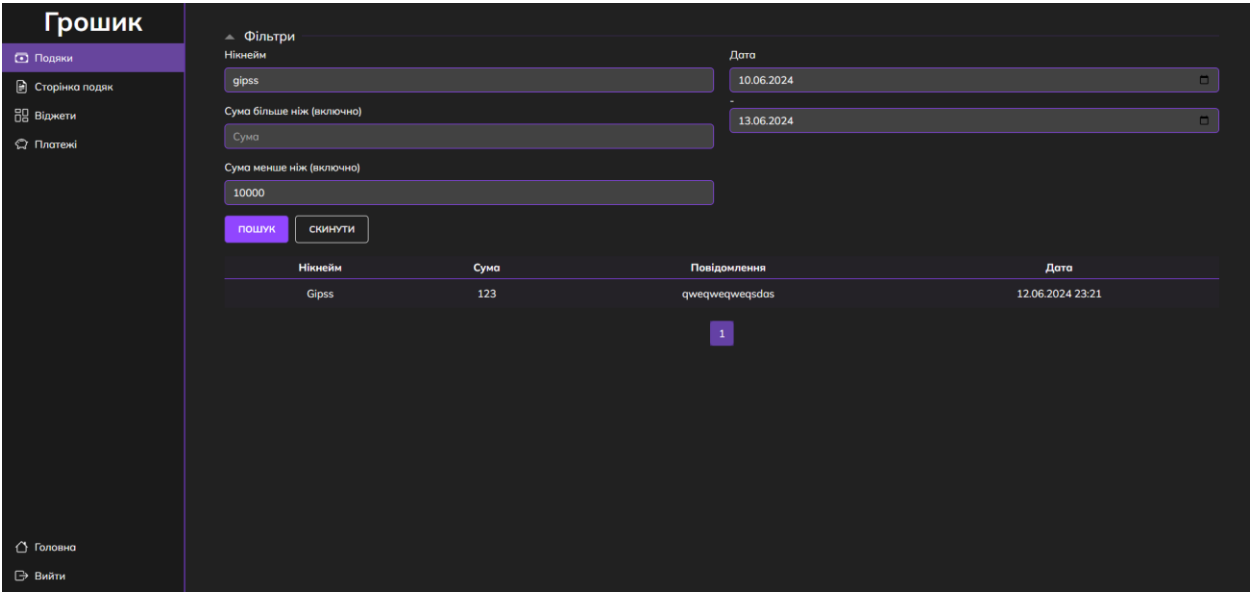


Рисунок 3.8 – Сторінка отриманих донатів із застосуванням фільтрації по декількох полях

Також на цій сторінці у стрімера є можливість переглянути детальну інформацію подяки, якщо повідомлення є занадто великим (рис. 3.9).

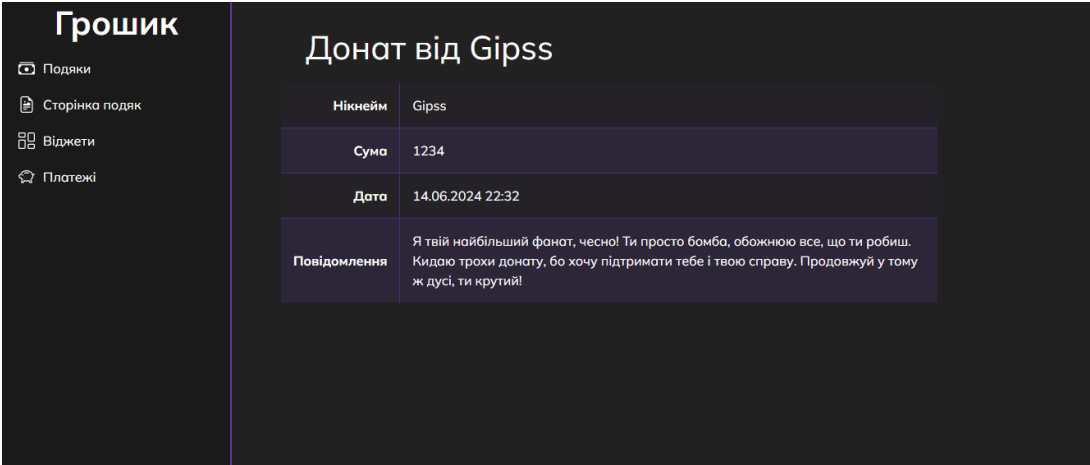


Рисунок 3.9 – Детальна інформація про донат

Наступної у меню йде інструмент для редагування конфігурацій сторінки подяк (рис. 3.10). Вона містить в собі посилання на працюючу сторінку і форму з полями для введення даних для певних властивостей сторінки подяки. Також у правій частині екрану знаходиться блок для попереднього огляду майбутньої сторінки, зміни на ній відобразяться одразу після збереження даних (рис. 3.11).

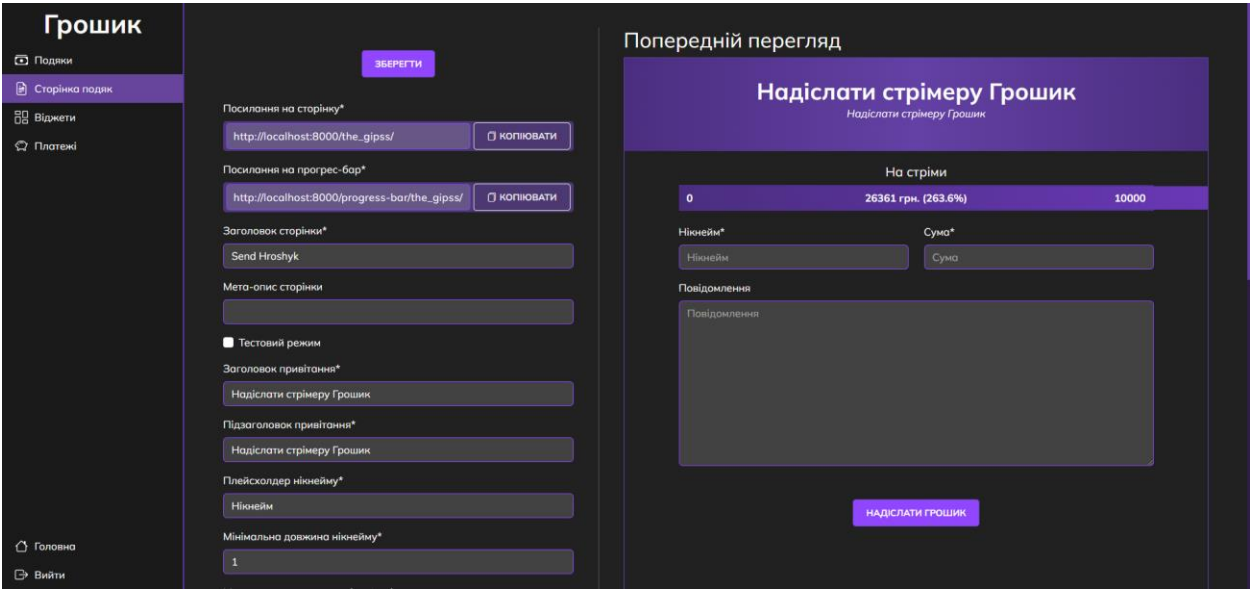


Рисунок 3.10 – Налаштування сторінки подяк

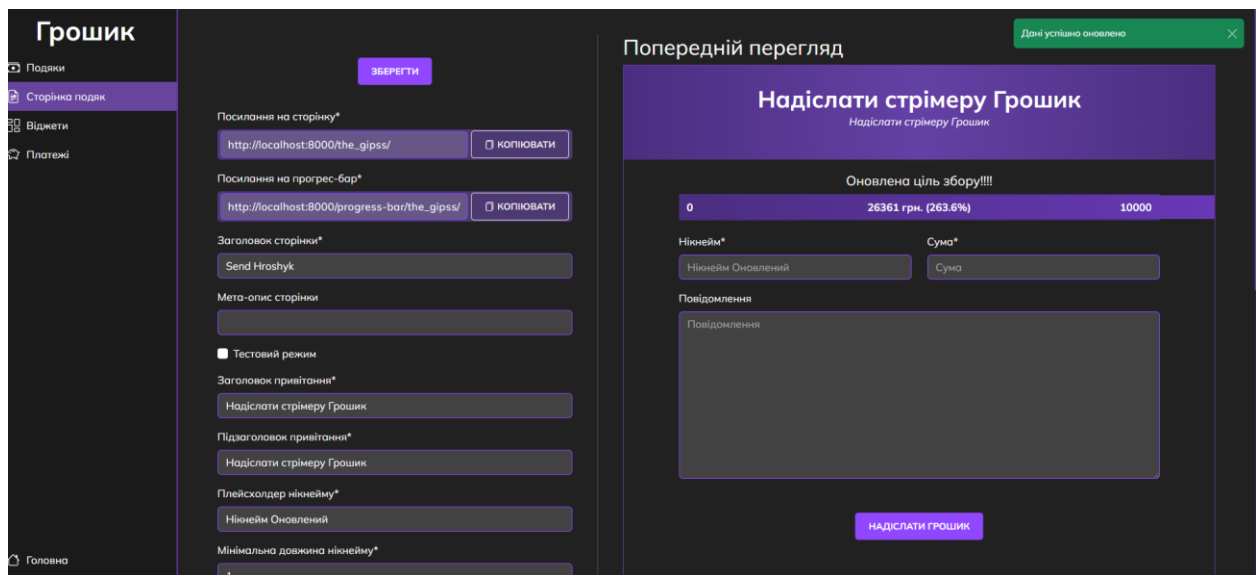


Рисунок 3.11 – Оновлення даних сторінки подяк

Також тут є можливість скопіювати посилання на сторінку подяк, яку будуть бачити глядачі. Після натискання на кнопку, з'являється відповідне повідомлення (рис. 3.12).

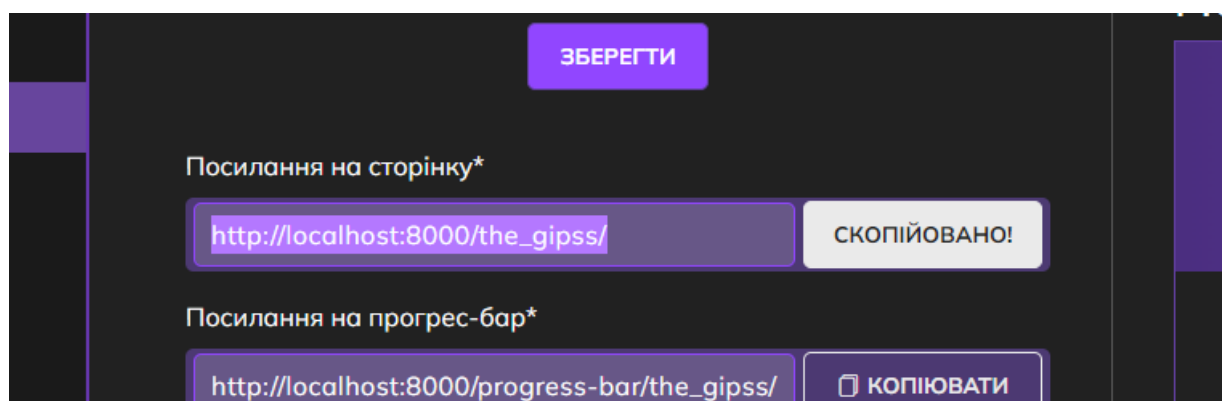


Рисунок 3.12 – Копіювання посилання на сторінку подяк

Наступною є сторінка створення та редагування налаштувань для віджетів (рис. 3.13). Вона є важливою ланкою платформи, оскільки гнучкість конфігурації віджетів є дуже важливою для стрімерів.

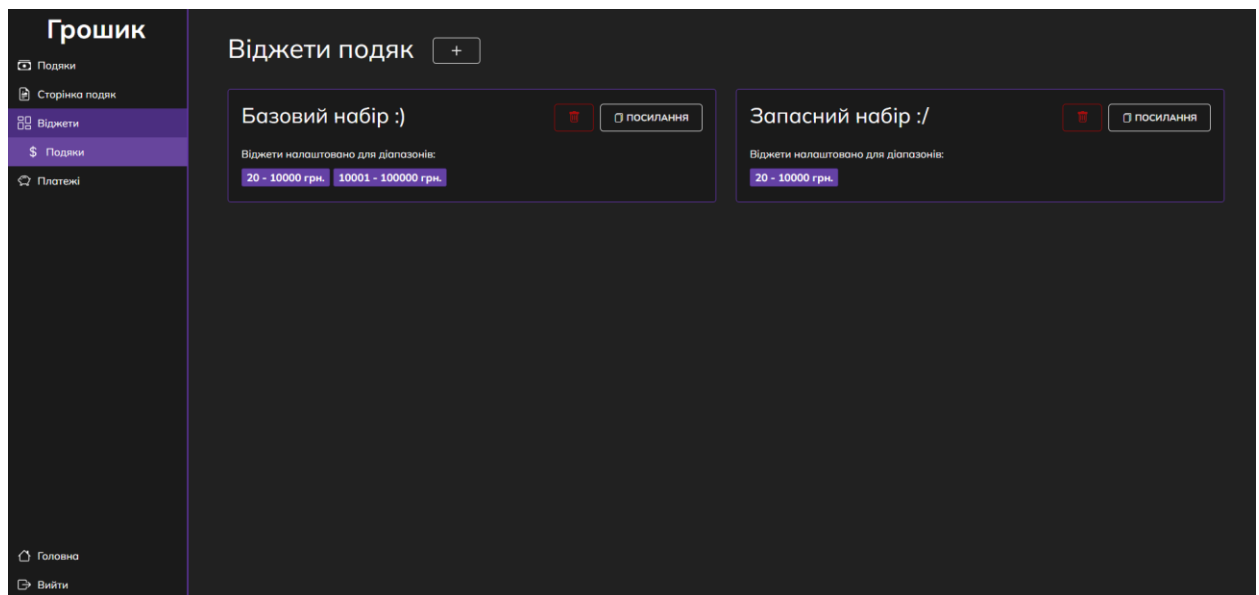


Рисунок 3.13 – Сторінка налаштувань віджетів

Звідси можна перейти на сторінку створення нового набору налаштувань, де потрібно обрати назву, а також ввести дані налаштування для першого віджету в цьому наборі (рис. 3.14).

Рисунок 3.14 – Створення нового набору віджетів

В результаті, набір конфігурацій віджетів, який щойно був створений буде виглядати приблизно так, як на рисунку 3.15.

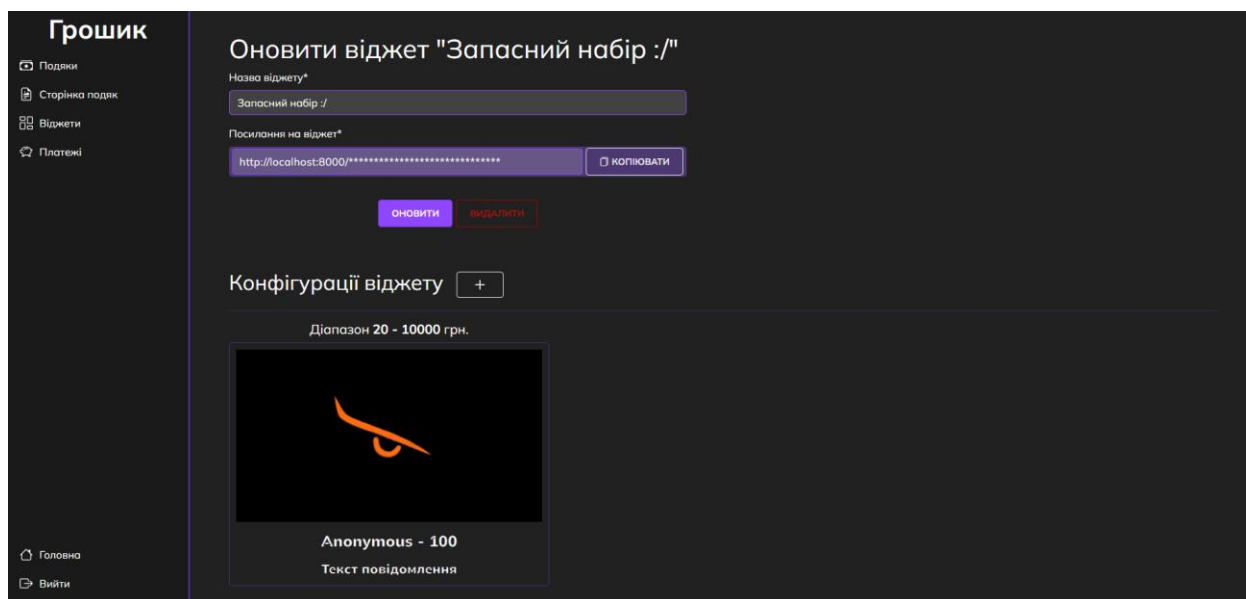


Рисунок 3.15 – Новий набір конфігурацій віджетів

На цій сторінці можна оновити назву набору, видалити його, створити нову конфігурацію або оновити її дані. При спробі видалення платформа запитає чи точно ви хочете видалити це налаштування для запобігання випадковому видаленню (рис. 3.16). Таку перевірку надалі матимуть усі кнопки видалення.

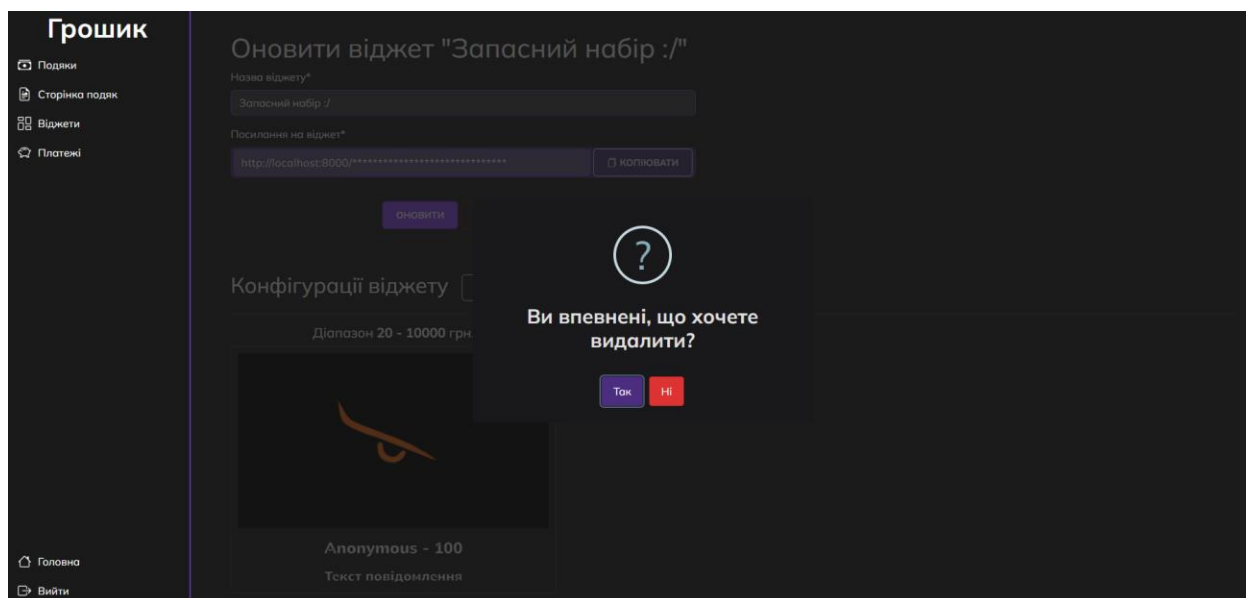


Рисунок 3.16 – Спроба видалення набору конфігурацій

З цієї сторінки також можна перейти на форму створення нової конфігурації (3.17). Вона містить ті ж поля, що і налаштування базового віджету при створенні набору.

Грошик

Подяки

Сторінка подяк

Віджети

Платежі

Головна

Вийти

Створити конфігурацію віджета

Зображення*

Вибрати файл

Screenshot 2023-12-18 205120.png

Звук*

Вибрати файл

audio_2024-06-12-23-20-53.ogg

Формат заголовку*

Глядач {nickname} задонатив {amount} гривень!

Мінімальна сума для відображення*

10001

Максимальна сума для відображення*

100000

Розмір шрифту заголовку*

24

Колір заголовку*

Колір тіні заголовку*

Розмір шрифту повідомлення*

20

Колір повідомлення*

Рисунок 3.17 – Створення нової конфігурації віджета

Після цього стрімер переадресовується на сторінку детальної інформації та редагування створеної конфігурації віджета (рис. 3.18). Тут він може оновити всі дані, змінити колір чи вміст. Також на сторінці набору конфігурацій з’явився новий блок з попереднім оглядом цього віджета (рис. 3.19).

Грошик

Подяки

Сторінка подяк

Віджети

Платежі

Головна

Вийти

Оновити конфігурацію віджета

Зображення*

Наразі

mediationation_widget/images/cklean_nofns

Вибрати файл

Файл не вибрано

Звук*

Наразі

mediationation_widget/sounds/audio_2024-06-12-23-20-53_14x9t.ogg

Вибрати файл

Файл не вибрано

Формат заголовку*

Глядач {nickname} задонатив {amount} гривень!

Мінімальна сума для відображення*

10001

Максимальна сума для відображення*

100000

Розмір шрифту заголовку*

24

Колір заголовку*

Колір тіні заголовку*

Розмір шрифту повідомлення*

Глядач Anonymous задонатив 100 гривень!

Текст повідомлення

Рисунок 3.18 – Щойно створена конфігурація віджета

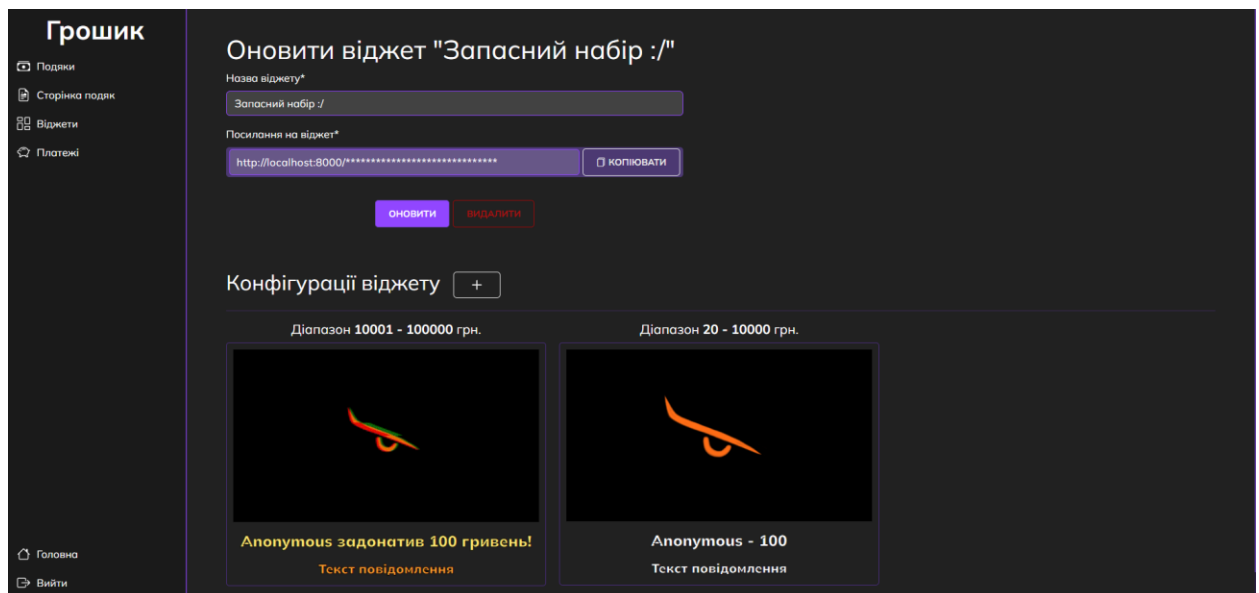


Рисунок 3.19 – Нова конфігурація віджета з’явилася в переліку
Після успішного видалення цей блок так само зникне звідси (рис. 3.20)

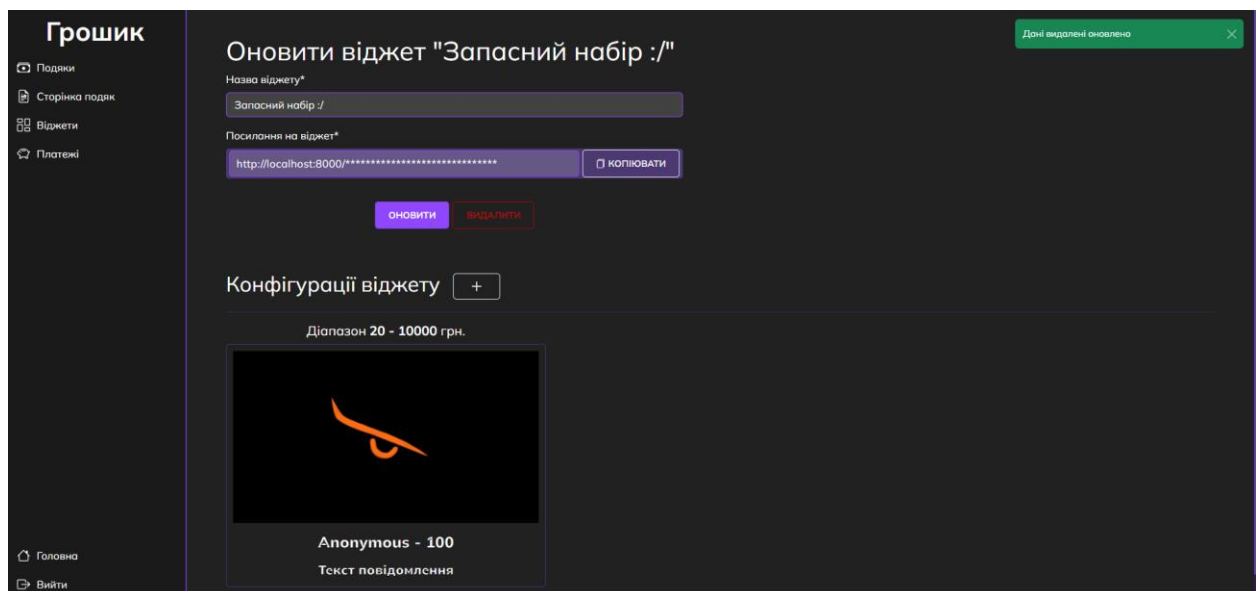


Рисунок 3.20 – Успішне видалення конфігурації

Важливою частиною також є сторінка створення заявок на виплату зібраних коштів (рис. 3.21). У таблиці відображаються створені стрімером заявки на весь час та їхні статуси. Після створення нової заявки вона одразу відобразиться у переліку зі статусом «Створено» (рис. 3.21).

Грошик

Подяки

Сторінка подяк

Віджети

Платежі

Головна

Вийти

Створити заявку на виплату

Сума*

Номер Картки*

4441 1234 4321 1234

ІПН/РНОКПП*

1234567890

СТВОРИТИ

ID	Номер Картки	Сума	Статус
d6da8723-8593-4a14-b42d-bc23ee63d812	4441114442677736	10000,00	Створено
6dec5de5-30c5-49a4-8606-dbd4d8bdd6a5	4441114442677736	50,00	У роботі

Рисунок 3.21 – Сторінка створення заявок на виплату накопичених коштів

Грошик

Подяки

Сторінка подяк

Віджети

Платежі

Головна

Вийти

Створити заявку на виплату

Заявку успішно створено

Сума*

666

Номер Картки*

4441114442677736

ІПН/РНОКПП*

2435345345

СТВОРИТИ

ID	Номер Картки	Сума	Статус
a728f909-306d-454d-882a-9c28a578af09	4441114442677736	666,00	Створено
d6da8723-8593-4a14-b42d-bc23ee63d812	4441114442677736	10000,00	Створено
6dec5de5-30c5-49a4-8606-dbd4d8bdd6a5	4441114442677736	50,00	У роботі

Рисунок 3.22 – Створення заявки

Мабуть найважливішою частиною платформи є процес створення і оплати донату глядачем. Сторінку подяки, яку бачить користувач, зображено нижче (рис. 3.23). На ній він повинен ввести нікнейм та суму і за бажанням може ввести повідомлення. Після введення даних глядач переходить на сторінку оплати донату, що знаходиться на боці платіжної системи (рис. 3.24), і у випадку успішної сплати опиняється знову на сторінці подяки з оновленим рахунком збору (рис. 3.25).

Надіслати стрімеру Грошик

Надіслати стрімеру Грошик

Оновлена ціль збору!!!!

0

0 грн. (0.0%)

10000

Нікнейм*

Сума*

Gips5

100

Повідомлення

Привіт, я прийшов на обід!

НАДІСЛАТИ ГРОШИК

Рисунок 3.23 – Створення нового донату

QR-код для оплати

Використовуйте Privat24

LIQPAY»

Дані про оплату

Donation to a streamer

До сплати: 100.00 UAH

Pay

Оплатити через Google Pay

або

Карта

Приват24

Розум

Номер картки

4441 1144 4267 7736

ВІСА

Термін дії

10/24

CV2

...

Прізвище власника картки*

HISSA

Ім'я власника картки*

DMYTRO

☐ Відправити квитанцію на e-mail

Натискаючи на кнопку «Сплатити», ви підтверджуєте що ознайомлені з переліком інформації про послугу та приймаєте умови публічного договору

Сплатити 100.00 UAH

Рисунок 3.24 – Сторінка оплати на боці платіжної системи

Надіслати стрімеру Грошик

Надіслати стрімеру Грошик

Оновлена ціль збору!!!!

0

100 грн. (1.0%)

10000

Нікнейм*

Сума*

Нікнейм Оновлений

Сума

Повідомлення

Повідомлення

НАДІСЛАТИ ГРОШИК

Рисунок 3.25 – Повернення на сторінку подяки з оновленим рахунком збору

Після цього, якщо стрімер використав інтеграцію зі стрімінговими програмами, на його трансляції з’явиться сповіщення про новий донат і віджет прогресу оновиться (рис. 3.26).

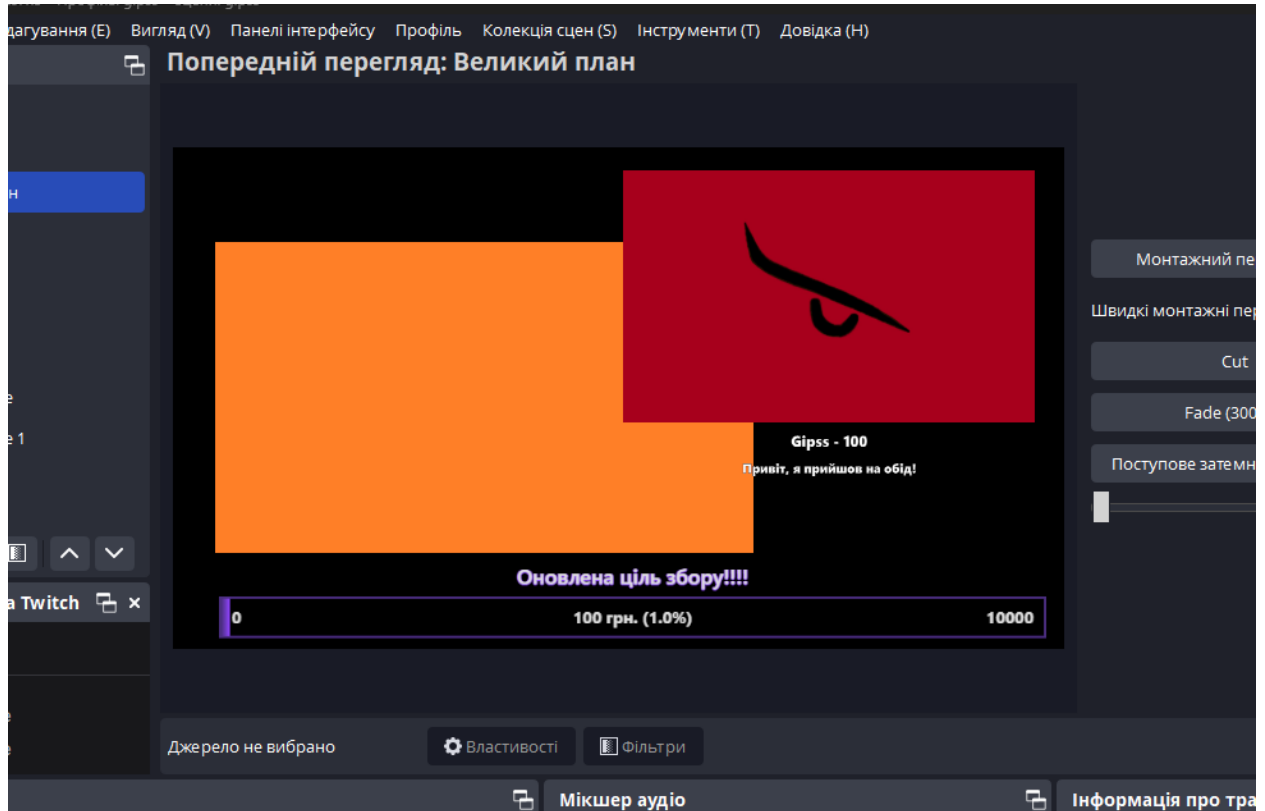


Рисунок 3.26 – Новий донат з’явився на трансляції

3.3 Тестування роботи програмного продукту

Розробка програмного забезпечення завжди супроводжується можливістю виникнення помилок. Саме тому тестування є надзвичайно важливим для забезпечення високої якості програмного продукту. Під час тестування відбувається аналіз різних аспектів програми, зокрема правильність обробки вхідних та вихідних даних, коректність виконання функцій та алгоритмів, а також обробка помилок і відновлення після збоїв. Основна мета тестування полягає у виявленні дефектів та помилок у програмному забезпеченні, що дозволяє усунути їх до виходу продукту на ринок або його впровадження в експлуатацію.

Для проведення тестування спочатку створюються "тест-кейси" (Test-Cases [22]). Тест-кейс – це документ, який визначає певні дії або умови для перевірки конкретної функціональності програмної системи, яка розробляється.

Нижче будуть перелічені тест-кейси для платформи для подяк стрімерам на Twitch.

В першу чергу необхідно перевірити чи стрімер має можливість реєструватися за допомогою Twitch. При першому вході до системи для нього повинен бути створений користувач і прив'язана до нього сторінка подяк. Опис цього сценарію представлений у таблиці 3.1.

Таблиця 3.1

Test Case 1

Опис сценарію	Кроки сценарію	Тестові дані	Очікуваний результат	Фактичний результат	Статус
Перевірка реєстрації за допомогою акаунту Twitch	1. Перейти на домашню сторінку. 2. Натиснути на кнопку реєстрації. 3. Підтвердити реєстрацію	Існуючий обліковий запис Twitch.	Користувач переадресовується до панелі керування і для нього була створення сторінка подяк.	Очікуваний	Успішно

Також необхідно перевірити сценарій автентифікації користувача за допомогою Twitch до вже існуючого акаунту, оскільки тоді вже не потрібно буде створювати запис в базі даних і сторінку подяк. Цей тестовий сценарій описано в таблиці 3.2.

Таблиця 3.2

Test Case 2

Опис сценарію	Кроки сценарію	Тестові дані	Очікуваний результат	Фактичний результат	Статус
Перевірка автентифікації за допомогою акаунту Twitch	1. Перейти на домашню сторінку. 2. Натиснути на кнопку входу.	1. Існуючий обліковий запис Twitch. 2. Бути зареєстрованим на платформі.	Користувач переадресовується до панелі керування, нових користувача і сторінку подяки не створено.	Очікуваний	Успішно

Також необхідно перевірити можливість стрімера перегляди отримані подяки та їх історію, шукати конкретні донати та бачити подяки за певний час. Цей сценарій описано в таблиці 3.3.

Таблиця 3.3

Test Case 3

Опис сценарію	Кроки сценарію	Тестові дані	Очікуваний результат	Фактичний результат	Статус
Пошук отриманих подяк за нікнеймом глядача, який їх надіслав	1. Перейти на сторінку перегляду подяк. 2. Ввести нікнейм глядача. 3. Натиснути на кнопку пошуку.	1. Автентифікований обліковий запис. 2. Нікнейм глядача - Gips.	На сторінці у переліку будуть відображені лише подяки, нікнейм яких містять у собі «Gips» незалежно від реєстру.	Очікуваний	Успішно

Наступним кроком є перевірка можливості створення нового набору налаштувань і базового віджета. Це є важливим пунктом, оскільки без цього на трансляції буде відсутня взаємодія глядачів зі стрімером. Цей тестовий сценарій описаний в таблиці 3.4.

Таблиця 3.4

Test Case 4

Опис сценарію	Кроки сценарію	Тестові дані	Очікуваний результат	Фактичний результат	Статус
Створення набору налаштувань з базовим віджетом	1. Перейти на сторінку віджетів. 2. Натиснути на кнопку створення нового набору налаштувань. 3. Ввести тестові дані у форму. 4. Натиснути на кнопку підтвердження.	1. Автентифікований обліковий запис. 2. Налаштування для набору конфігурацій.	Новий набір створено разом з базовим віджетом для нього.	Очікуваний	Успішно

Для стрімера дуже важливим є можливість отримання накопичених ним коштів на його особисту картку. Для цього йому необхідно створити заявку на

виплату на сторінці платежів. Тестування цього функціоналу описано в таблиці 3.5.

Таблиця 3.5

Test Case 5

Опис сценарію	Кроки сценарію	Тестові дані	Очікуваний результат	Фактичний результат	Статус
Створення заявки на виплату накопичених коштів	1. Перейти на сторінку платежів. 2. Ввести тестові дані. 3. Натиснути на кнопку створення.	1. Автентифікований обліковий запис. 2. Сума, номер картки стрімера і його ІПН/РНОКПП.	Нову заявку створено і в таблиці можна побачити її статус.	Очікуваний	Успішно

Основним функціоналом платформи є можливість глядачів надсилати подяки стрімерам, яких вони хочуть підтримати, з подальшим відображенням донату на трансляції. Цей тестовий сценарій описано в таблиці 3.6.

Таблиця 3.6

Test Case 6

Опис сценарію	Кроки сценарію	Тестові дані	Очікуваний результат	Фактичний результат	Статус
Надсилання подяки стрімеру	1. Перейти на сторінку подяки. 2. Ввести нікнейм, суму і повідомлення. 3. Ввести платіжну інформацію і підтвердити платіж.	1. Нікнейм, сума і повідомлення для подяки. 2. Платіжна інформація глядача.	Новий донат створено і після його успішної оплати він відобразиться на трансляції.	Очікуваний	Успішно

Висновки до третього розділу

В процесі виконання третього розділу було розглянуто схему розгортання платформи, а також описано взаємодію її модулів між собою.

Також було описано і продемонстровано на рисунках процес роботи платформи та які саме функції присутні на ній. Стрімер має можливість

переглядати отримані донати, фільтрувати їх, редагувати вміст на характеристики сторінки подяки, створювати різноманітні конфігурації віджетів для різних діапазонів суми донатів та створювати заявки на отримання накопичених коштів. А глядач має можливість створювати і надсилати подяки з їх подальшою успішною оплатою.

В цьому розділі також було проведено тестування найбільш головних функцій системи за допомогою Test Cases. В результаті платформа продемонструвала успішне проходження випробуваних тестових сценаріїв.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було розроблено та реалізовано платформу для подяк стрімерам на Twitch з метою покращення взаємодії між авторами та аудиторією. Під час дослідження предметної області були визначені основні проблеми, які повинна вирішувати платформа, а також сформульовано конкретні завдання. Проведено аналіз аналогічних систем і платформ, визначено їхні переваги та недоліки, на основі чого сформовано першочергові цілі продукту. Для реалізації проекту обрано відповідні технології: Python і фреймворк Django для серверної частини, PostgreSQL як базу даних, та HTML, CSS і JavaScript-бібліотеку HTMX для клієнтської частини.

Проектування платформи включало аналіз варіантів використання програмного продукту та побудову діаграм діяльності користувачів. Визначено функціональні та нефункціональні вимоги, що сприяло плануванню структури коду та модулів. Було спроектовано структуру бази даних з 6 таблиць, що забезпечують зберігання необхідної інформації стрімерів, адміністраторів, подяк, конфігурацій сторінок та віджетів, а також заявок на виплату коштів. Також було розглянуто код серверної частини програмного продукту, зокрема модулі, що відповідають за інтеграцію з платіжної системою і стрімінговими програмами. Було знайдено і обґрунтовано рішення певних проблем, що виникали під час написання коду.

Описано схему розгортання платформи та взаємодію її модулів. Демонстрація процесу роботи платформи включає можливості стрімера переглядати, фільтрувати донати, редагувати характеристики сторінки подяки, створювати конфігурації віджетів та заявки на виплату коштів. Глядачі можуть створювати і надсилати подяки з подальшою оплатою. Проведено тестування основних функцій системи за допомогою Test Cases, результати якого показали успішне виконання всіх тестових сценаріїв.

Результатом роботи є готова платформа для подяк стрімерам на Twitch, яка забезпечує сучасний та зрозумілий інтерфейс, адаптивність, підтримку

англійської мови, зручну кастомізацію віджетів, нижчу комісію порівняно з конкурентами, відсутність обмежень на дохід, інтеграцію зі стрімінговими програмами та стабільну роботу. У майбутньому розвитку системи можлива реалізація додаткових функцій для покращення користувацького досвіду.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Twitch [Електронний ресурс] – Режим доступу до ресурсу: <https://www.twitch.tv/>.
2. Документація Python [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.python.org/3/>.
3. Документація Django [Електронний ресурс] – Режим доступу до ресурсу: <https://www.djangoproject.com/>.
4. Підручник з HTML [Електронний ресурс] – Режим доступу до ресурсу: <https://w3schoolsua.github.io/index.html>.
5. CSS довідник [Електронний ресурс] – Режим доступу до ресурсу: <https://css.in.ua/>.
6. Документація JavaScript [Електронний ресурс] – Режим доступу до ресурсу: <https://devdocs.io/javascript/>.
7. PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: <https://www.postgresql.org/>.
8. OBS [Електронний ресурс] – Режим доступу до ресурсу: <https://obsproject.com/uk>.
9. Diaka [Електронний ресурс] – Режим доступу до ресурсу: <https://diaka.ua/>.
10. Donatello [Електронний ресурс] – Режим доступу до ресурсу: <https://donatello.to/>.
11. Monobank [Електронний ресурс] – Режим доступу до ресурсу: <https://monobank.ua/>.
12. MVC Framework Introduction [Електронний ресурс] – Режим доступу до ресурсу: <https://www.geeksforgeeks.org/mvc-framework-introduction/>.
13. Документація HTMX [Електронний ресурс] – Режим доступу до ресурсу: <https://htmx.org/>.
14. SPA (Single-page application) [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/docs/Glossary/SPA>.
15. Патерн Стратегія [Електронний ресурс] – Режим доступу до ресурсу: <https://refactoring.guru/uk/design-patterns/strategy>.

16. PyCharm [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.jetbrains.com/pycharm/>.
17. Документація Bootstrap [Електронний ресурс] – Режим доступу до ресурсу:
<https://getbootstrap.com/>.
18. PostgreSQL Advantages and Disadvantages [Електронний ресурс] – Режим доступу до ресурсу: <https://www.aalpha.net/blog/pros-and-cons-of-using-postgresql-for-application-development/>.
19. PostgreSQL vs MySQL: The Critical Differences [Електронний ресурс] – Режим доступу до ресурсу: <https://www.integrate.io/blog/postgresql-vs-mysql-which-one-is-better-for-your-use-case/>.
20. LiqPay [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.liqpay.ua/uk>.
21. Django, Translation [Електронний ресурс] – Режим доступу до ресурсу:
<https://docs.djangoproject.com/en/5.0/topics/i18n/translation/>.
22. What is a Test Case [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.techtarget.com/searchsoftwarequality/definition/test-case>.

ДОДАТКИ

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	
						67

```

from urllib.parse import urljoin

from django.contrib.auth.mixins import LoginRequiredMixin
from django.http import Http404, QueryDict
from django.shortcuts import get_object_or_404, redirect
from django.urls import reverse
from django.views import View
from django.views.generic import CreateView, DetailView
from django.views.generic.detail import SingleObjectMixin
from django_filters.views import FilterView
from liqpay import LiqPay

from accounts.models import Streamer
from donation_page.models import DonationPage
from donations.filters import DonationFilter
from donations.forms import DonationForm, FullDonationForm
from donations.models import Donation
from donations.utils import donation_add_validation
from hroshyk import settings

class DonationsListView(LoginRequiredMixin, FilterView):
    model = Donation
    paginate_by = 25
    template_name = "donations/donation_list.html"
    context_object_name = "donations"
    ordering = "-created_at"
    filterset_class = DonationFilter

    def get(self, request, *args, **kwargs):
        if request.htmx and not request.htmx.boosted:
            self.template_name = "donations/donation_list_table.html"
        elif request.htmx:
            self.template_name = "donations/donation_list_content.html"
        else:
            self.template_name = "donations/donation_list.html"
        return super().get(request, *args, **kwargs)

    def get_queryset(self):
        return super().get_queryset().filter(payed=True)

class DonationDetailView(LoginRequiredMixin, DetailView):
    model = Donation
    template_name = "donations/donation_detail.html"
    context_object_name = "donation"

    def get(self, request, *args, **kwargs):
        if request.htmx:
            self.template_name = "donations/donation_detail_content.html"
        else:
            self.template_name = "donations/donation_detail.html"
        return super().get(request, *args, **kwargs)

class ViewerDonationPageView(CreateView):
    template_name = "donation_page/viewer_donation_page.html"
    form_class = DonationForm
    object: Donation | None

    def get(self, request, *args, **kwargs):

```

```

        link = kwargs["link"]
        response = super().get(request, *args, **kwargs)
        response.context_data["donation_page"] = get_object_or_404(DonationPage,
page_link=link)
        form = response.context_data["form"]
        obj = response.context_data["donation_page"]
        donation_add_validation(form, obj)
        return response

    def post(self, request, *args, **kwargs):
        self.object = None
        form = FullDonationForm(**self.get_form_kwargs())
        result = self.form_valid(form) if form.is_valid() else
self.form_invalid(form)
        liqpay = LiqPay(settings.LIQPAY_PUBLIC_KEY, settings.LIQPAY_PRIVATE_KEY)
        request_data = liqpay._prepare_params(
            {
                "action": "pay",
                "amount": str(self.object.amount),
                "currency": "UAH",
                "description": "Donation to a streamer",
                "order_id": str(self.object.pk),
                "version": "3",
                "result_url": request.build_absolute_uri(
                    reverse(
                        "successful_donation",
                        kwargs={"pk": self.object.pk, "link":
self.object.streamer.donation_page.page_link},
                    )
                ),
            }
        )
        params = {"data": liqpay.data_to_sign(request_data)}
        params["signature"] = liqpay._make_signature(liqpay._private_key,
params["data"], liqpay._private_key)
        url = urljoin(liqpay._host, "3/checkout/") +
f"?data={params['data']}&signature={params['signature']}"
        return redirect(url)

    def get_form_kwargs(self):
        kwargs = super().get_form_kwargs()
        if self.request.method != "POST":
            return kwargs
        link = self.kwargs["link"]
        streamer = Streamer.objects.get(donation_page__page_link=link)
        if not streamer:
            raise Http404
        kwargs["data"] = QueryDict(self.request.POST.copy().urlencode(),
mutable=True)
        kwargs["data"]["streamer"] = str(streamer.id)
        return kwargs

class SuccessfulDonationView(SingleObjectMixin, View):
    model = Donation

    def get(self, request, *args, **kwargs):
        self.object = self.get_object()
        self.object.payed = True
        self.object.streamer.balance += self.object.amount
        self.object.streamer.donation_page.target_current_amount +=
self.object.amount
        self.object.save()
        self.object.streamer.save()

```

```
self.object.streamer.donation_page.save()
return redirect(reverse("viewer_donation_page", kwargs={"link":
kwargs.get("link")}))
```

```

from functools import reduce

from django.contrib import messages
from django.contrib.auth.mixins import LoginRequiredMixin
from django.db.models import Prefetch, Q
from django.http import HttpResponseForbidden, Http404, HttpResponse
from django.urls import reverse_lazy
from django.utils.translation import gettext
from django.shortcuts import render
from django.views.generic import ListView, UpdateView, FormView, DeleteView,
DetailView
from django_htmx.http import HttpResponseLocation

from donations.models import Donation
from widgets.forms import (
    DonationWidgetForm,
    CreateDonationWidgetForm,
    DonationWidgetConfigFormSet,
    DonationWidgetConfigForm,
)
from widgets.models import DonationWidget, DonationWidgetConfig

class DonationWidgetListView(LoginRequiredMixin, ListView):
    template_name = "widgets/donation_widget/donation_widget_list.html"
    model = DonationWidget
    context_object_name = "donation_widgets"
    ordering = ["id"]

    def get_queryset(self):
        return self.model.objects.filter(user=self.request.user).order_by("id")

    def get(self, request, *args, **kwargs):
        if request.htmx:
            self.template_name =
"widgets/donation_widget/donation_widget_list_content.html"
        else:
            self.template_name =
"widgets/donation_widget/donation_widget_list.html"
        return super().get(request, *args, **kwargs)

class DonationWidgetCreateView(LoginRequiredMixin, FormView):
    template_name = "widgets/donation_widget/donation_widget_create.html"
    form_class = CreateDonationWidgetForm
    success_url = reverse_lazy("donation_widgets_list")

    def get(self, request, *args, **kwargs):
        if request.htmx:
            self.template_name =
"widgets/donation_widget/donation_widget_create_content.html"
        else:
            self.template_name =
"widgets/donation_widget/donation_widget_create.html"
        if request.user.donation_widgets.count() >= 3:
            messages.error(request, gettext("You cannot create more than 3
widgets"))
        return HttpResponseForbidden()
        return super().get(request, *args, **kwargs)

    def get_context_data(self, **kwargs):

```

```

        context = super().get_context_data(**kwargs)
        if self.request.POST:
            context["formset"] = DonationWidgetConfigFormSet(
                self.request.POST,
                self.request.FILES,
                instance=DonationWidget(),
            )
        else:
            context["formset"] =
DonationWidgetConfigFormSet(instance=DonationWidget())
        return context

    def form_valid(self, form):
        formset = DonationWidgetConfigFormSet(self.request.POST,
self.request.FILES, instance=DonationWidget())
        if formset.is_valid():
            form.instance.user = self.request.user
            donation_widget = form.save()
            instances = formset.save(commit=False)
            for instance in instances:
                instance.donation_widget = donation_widget
                instance.save()

            return HttpResponseRedirect(self.get_success_url(), target="main")

        return self.render_to_response(self.get_context_data(form=form,
formset=formset))

class DonationWidgetUpdateView(LoginRequiredMixin, UpdateView):
    template_name = "widgets/donation_widget/donation_widget_update.html"
    model = DonationWidget
    context_object_name = "donation_widget"
    form_class = DonationWidgetForm

    def get(self, request, *args, **kwargs):
        if request.htmx:
            self.template_name =
"widgets/donation_widget/donation_widget_update_content.html"
        else:
            self.template_name =
"widgets/donation_widget/donation_widget_update.html"
        return super().get(request, *args, **kwargs)

    def get_queryset(self):
        return self.model.objects.prefetch_related(
            Prefetch("configs",
                queryset=DonationWidgetConfig.objects.order_by("-id")),
        )

    def form_valid(self, form):
        if self.object.user != self.request.user:
            return HttpResponseRedirect()
        self.object = form.save()
        messages.success(self.request, gettext("Data updated successfully"))

        if self.request.htmx:
            self.template_name =
"widgets/donation_widget/donation_widget_update_content.html"
        else:
            self.template_name =
"widgets/donation_widget/donation_widget_update.html"

        return self.render_to_response(self.get_context_data(form=form,

```

```

donation_widget=self.object))

class DonationWidgetDeleteView(LoginRequiredMixin, DeleteView):
    model = DonationWidget
    success_url = reverse_lazy("donation_widgets_list")

    def form_valid(self, form):
        if self.object.user != self.request.user:
            return HttpResponseForbidden()
        self.object.delete()
        return HttpResponseRedirect(self.success_url, target="main")

class DonationWidgetConfigCreateView(LoginRequiredMixin, FormView):
    form_class = DonationWidgetConfigForm
    template_name = "widgets/donation_widget/donation_widget_config_create.html"

    def get(self, request, *args, **kwargs):
        if request.htmx:
            self.template_name =
"widgets/donation_widget/donation_widget_config_create_content.html"
        else:
            self.template_name =
"widgets/donation_widget/donation_widget_config_create.html"
        return super().get(request, *args, **kwargs)

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        donation_widget_id = kwargs.get("donation_widget_id") or
int(self.request.GET.get("donation_widget_id"))
        if not donation_widget_id or not
DonationWidget.objects.filter(pk=donation_widget_id).exists():
            return HttpResponse(
                context["donation_widget_id"] = donation_widget_id
            return context

    def form_valid(self, form):
        self.object = form.save(commit=False)
        donation_widget_id = int(self.request.GET.get("donation_widget_id"))
        if not donation_widget_id or not
DonationWidget.objects.filter(pk=donation_widget_id).exists():
            return HttpResponse(
                self.object.donation_widget_id = donation_widget_id
                self.object.save()
                success_url = reverse_lazy("donation_widgets_config_update",
kwargs={"pk": self.object.pk})
                return HttpResponseRedirect(success_url, target="main")

class DonationWidgetConfigUpdateView(LoginRequiredMixin, UpdateView):
    model = DonationWidgetConfig
    form_class = DonationWidgetConfigForm
    context_object_name = "donation_widget_config"
    template_name = "widgets/donation_widget/donation_widget_config_update.html"

    def get(self, request, *args, **kwargs):
        if request.htmx:
            self.template_name =
"widgets/donation_widget/donation_widget_config_update_content.html"
        else:
            self.template_name =
"widgets/donation_widget/donation_widget_config_update.html"
        return super().get(request, *args, **kwargs)

```

```

def form_valid(self, form):
    if self.object.donation_widget.user != self.request.user:
        return HttpResponseRedirect()
    self.object = form.save()
    messages.success(self.request, gettext("Data updated successfully"))

    if self.request.htmx:
        self.template_name =
"widgets/donation_widget/donation_widget_config_update_content.html"
    else:
        self.template_name =
"widgets/donation_widget/donation_widget_config_update.html"

    return self.render_to_response(self.get_context_data(form=form,
donation_widget_config=self.object))

class DonationWidgetConfigDeleteView(LoginRequiredMixin, DeleteView):
    model = DonationWidgetConfig

    def form_valid(self, form):
        if self.object.donation_widget.user != self.request.user:
            return HttpResponseRedirect()
        donation_widget_id = self.object.donation_widget_id
        self.object.delete()
        messages.success(self.request, gettext("Data deleted successfully"))
        success_url = reverse_lazy("donation_widgets_update", kwargs={"pk":
donation_widget_id})
        return HttpResponseRedirect(success_url, target="main")

class DonationAlertView(DetailView):
    model = DonationWidget
    template_name = "widgets/donation_widget/donation_alert.html"

    def get(self, request, *args, **kwargs):
        if not request.htmx:
            return super().get(request, *args, **kwargs)

        self.object = self.get_object()
        ranges = {config.pk: (config.min_amount, config.max_amount) for config
in self.object.configs.all()}

        donation_to_show = (
            Donation.objects.filter(streamer_id=self.object.user_id,
shown=False, payed=True)
            .filter(reduce(lambda x, y: x | y, [Q(amount__range=range_) for
range_ in ranges.values()]))
            .order_by("created_at")
            .first()
        )

        if not donation_to_show:
            return HttpResponseRedirect("")

        config_id = next(
            (id_ for id_, range_ in ranges.items() if range_[0] <=
donation_to_show.amount <= range_[1]),
            None,
        )

        config = next((config for config in self.object.configs.all() if
config.pk == config_id), None)

```

```

    if not config:
        return HttpResponse("No config found for this donation amount")

    donation_to_show.shown = True
    donation_to_show.save()

    template_name = "widgets/donation_widget/donation_alert_content.html"

    return render(request, template_name, {"donation": donation_to_show,
"config": config})

def get_object(self, queryset=None):
    try:
        return self.model.objects.prefetch_related(
            Prefetch("configs",
queryset=DonationWidgetConfig.objects.order_by("id"))
        ).get(link_identifier=self.kwargs.get("widget_identifier"))
    except self.model.DoesNotExist as e:
        raise Http404() from e

```