

Центральноукраїнський національний технічний університет
 Механіко-технологічний факультет
 Кафедра кібербезпеки та програмного забезпечення

“Допущено до захисту”
 Завідувач кафедри кібербезпеки
 та програмного забезпечення
 д.т.н., професор
 _____ Олексій СМІРНОВ
 “_____” _____ 2026 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
 на тему

“Програмне забезпечення системи конкурентного розподілу задач між учасниками виробництва відеоконтенту з використанням аукціонних механізмів та оцінюванням ефективності”

Виконав здобувач вищої освіти
 IV курсу, групи КН-22
 ОПП “Комп’ютерні науки”
 спеціальність 122 “Комп’ютерні науки”
 _____ Пащенко Д. В.
 “_____” _____ 2026 р.

Керівник проекту
 кандидат технічних наук, доцент
 _____ Дресєв О. М.
 “_____” _____ 2026 р.
 Рецензент _____

Центральноукраїнський національний технічний університет
 Факультет Механіко-технологічний
 Кафедра Кібербезпеки та програмного забезпечення
 Рівень вищої освіти бакалавр
 Галузь знань 12 “Інформаційні технології”
 Спеціальність 122 “Комп’ютерні науки”
 Освітньо-професійна (освітньо-наукова) програма “Комп’ютерні науки”

ЗАТВЕРДЖУЮ

Завідувач кафедри

д.т.н., проф.

Олексій СМІРНОВ

“ ” 2026 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Пащенко Данила Віталійовича

(прізвище, ім'я, по батькові)

1. Тема роботи Програмне забезпечення системи конкурентного розподілу задач між учасниками виробництва відеоконтенту з використанням аукціонних механізмів та оцінюванням ефективності

2. Керівник роботи Дресев Олександр Миколайович, канд. техн. наук, доцент
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “06” лютого 2026 року № 116 02

3. Строк подання роботи до захисту 22.05.2026 р.

4. Мета та завдання випускної кваліфікаційної роботи Метою роботи є розробка алгоритму підтримки сервісу колективно-конкурентного планування створення відео-контенту

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.

2. Перегляд аналогічних існуючих систем.

3. Опис і обґрунтування проектних рішень.

4. Етапи програмування системи.

5. Впровадження системи в промислову експлуатацію.

6. Висновки.

6. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

Структурна схема системи 1 аркуш

Функціональна схема системи 1 аркуш

Діаграма процесів 1 аркуш

Блок-схема алгоритму роботи додатку 2 аркуш

7. Дата видачі завдання “ 3 ” грудня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2026 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2026 р.	
3.	Розробка моделі компонента	20.03.2026 р.	
4.	Розробка структур даних	25.03.2026 р.	
5.	Розробка алгоритмів зв'язку та відображення	30.03.2026 р.	
6.	Програмування алгоритмів	10.04.2026 р.	
7.	Оформлення ПЗ	17.05.2026 р.	
8.	Попередній захист роботи	09.06.2026 р.	

Дата видачі завдання

“ 3 ” грудня 2025 р.

Підпис керівника

_____ (прізвище та ініціали)

Завдання прийнято до виконання

“ 3 ” грудня 2025 р.

Підпис здобувача

_____ (прізвище та ініціали)

АНОТАЦІЯ

Пашенко Д. В. Розробка алгоритму підтримки сервісу колективно-конкурентного планування створення відео-контенту. 122 Комп'ютерні науки. Центральноукраїнський Національний Технічний Університет. Кропивницький. 2026.

В випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено веб-додаток, який призначений для алгоритму підтримки сервісу колективно-конкурентного планування створення відео-контенту.

Метою роботи є веб-додаток, який реалізує алгоритм підтримки сервісу колективно-конкурентного планування створення відео-контенту.

Результат роботи – веб-додаток алгоритму підтримки сервісу колективно-конкурентного планування створення відео-контенту.

В процесі роботи над програмною моделлю виконано аналіз існуючих програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведено інструкції по роботі з програмними засобами.

Веб-додаток може використовуватись на ПК з будь-якою ОС.

Програму розроблено в середовищі NodeJS, з використанням фреймворку React.

Ключові слова: комп'ютерні науки, веб-розробка, Frontend, Backend, веб-додаток, клієнт-серверна архітектура, відео-контент.

ABSTRACT

Pashchenko D. V. Development of an algorithm for supporting a service of collective-competitive planning for video content creation. 122 Computer Science. Central Ukrainian National Technical University. Kropyvnytskyi. 2026.

In the bachelor's qualification work, a web application intended for implementing an algorithm to support a service of collective-competitive planning for video content creation was developed.

The purpose of the work is to create a web application that implements an algorithm for supporting a service of collective-competitive planning for video content creation.

The result of the work is a web application implementing an algorithm for supporting a service of collective-competitive planning for video content creation.

During the development of the software model, an analysis of existing software solutions was carried out. All components of the developed software are fully described.

A user-friendly interface was developed. Instructions for working with the software tools are provided.

The web application can be used on personal computers with any operating system.

The program was developed in the NodeJS environment using the React framework.

Keywords: computer science, web development, Frontend, Backend, web application, client-server architecture, video content

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ ..	2
ВСТУП.....	3
1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ	6
1.1 Призначення системи.....	6
1.2 Область застосування	7
2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ	9
2.1 Огляд існуючих систем, технологій, архітектур та програмних рішень....	9
2.2 Обґрунтування вибору засобів для побудови системи та мови програмування	15
2.3 Розгорнута постановка задачі	18
3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ	19
3.1 Опис функціонування системи	19
3.2 Розробка структурної схеми.....	24
3.3 Розробка функціональної схеми	27
3.4 Розробка діаграми процесів	30
4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ ТА ЕКСПЕРЕМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ І ПРОГРАМНИХ РІШЕНЬ	33
4.1 Розробка блок-схем та опис алгоритмів функціонування системи	33
4.2 Захист розробленого програмного забезпечення.....	45
5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ	46
6 ОСНОВНІ ВИСНОВКИ.....	49
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	51

					ВКРБ-122.26.0023.00.00.ПЗ			
Зм.	Арк.	№ докум.	Підпис	Дата	Програмне забезпечення системи конкурентного розподілу задач між учасниками виробництва відеоконтенту з використанням аукціонних механізмів та оцінюванням ефективності Літ. Аркуш Аркушів Б I 55 ЦНТУ КН-22			
Розроб.		Пашенко Д.В.						
Перевір.		Дресев О.М.						
Н. Контр.		Коваленко А. С.						
Затв.		Смірнов О. А.						

ПЕРЕЛІК УСЛОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

OTT	–	Over-the-top
IPTV	–	Internet Protocol Television
URL	–	Uniform Resource Locator
FIFO	–	First In, First Out
CSS	–	Cascading Style Sheets
SCSS	–	Sassy CSS
SQL	–	Structured Query Language
URI	–	Uniform Resource Identifier
HTTP(S)	–	Hyper Text Transfer Protocol (Secure)
JWT	–	JSON Web Token
API	–	Application Programming Interface
REST API	–	Representational State Transfer
TLS	–	Transport Layer Security
TCP	–	Transmission Control Protocol
IP	–	Internet Protocol
Хук	–	це спеціальна функція React, яка дозволяють підключитись до внутрішніх механізмів програми та фреймворку React.

ВСТУП

Актуальність теми. З розвитком інтернет-технологій відбулась фундаментальна зміна у сфері розваг: стрімінгові сервіси, такі як YouTube, Twitch та інші, стали потужними та домінуючими інструментами монетизації взаємодії між авторами контенту та тими, хто споживає контент – аудиторія автора.

В світі з кожним роком спостерігається стрімке збільшення попиту на OTT-платформи серед користувачів мережі Інтернет, пропонуючи доступ до будь-якого контенту в будь-якому місці в будь-який час на будь-якому пристрої за принципом “anywhere, anytime, anydevice”. В Україні станом на 2025 рік абсолютним лідером серед OTT-сервісів є YouTube з охопленням 63% користувачів, а у 2024-му частка OTT/IPTV у структурі телеперегляду становила 37% з темпами зростання на 9-10% в рік .

Сучасний ринок стрімінгу відходить від стандартної концепції перегляду контенту “дивись, що пропонують” до моделей гейміфікації, які збільшують залученість глядачів на 60%, за допомогою надання інтерактивності в процес взаємодії з контентом. Ринок гейміфікації, за прогнозами, сягне 30,7 млрд доларів до 2025 року.

В Україні в умовах повномасштабної війни особливо виросла сфера стрімінгу та попит на неї, через різкий запит на коментування політичних подій та розбору відеоматеріалів. Багато людей хочуть надати свою оцінку подіям, а також багато людей хочуть почути аналіз подій від лідерів думок. Це призвело до появи ніші “реакційного стрімінгу”, де автор у прямому ефірі переглядає та коментує контент, який йому надає його аудиторія. Але традиційні методи монетизації і вибору контенту не забезпечують глядачам ефективного механізму впливу на контент. Стрімеру, як правило, складно знайти підходящий контент для прямого ефіру, який б задовільнив його аудиторію та структурував б попит глядача.

Вим	Дир	№ докум	Підпис	Дата

ВКРБ-122.26.0023.00.00.ПЗ

Арк.

3

Тому створення зручного веб-додатку, який реалізує алгоритм підтримки сервісу колективно-конкурентного планування створення відео-контенту, який дозволять глядачам ефективно впливати на контент через систему диференційованих за ціною черг, є беззаперечно потрібним та актуальним рішенням. Це особливо актуально для контент-творців, які планують створювати та монетизувати свій “реакційний” контент.

Мета і задачі роботи. Метою роботи є розробка алгоритму підтримки сервісу колективно-конкурентного планування створення відео-контенту.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- Перегляд аналогічних існуючих систем, які реалізують алгоритм підтримки сервісу колективно-конкурентного планування створення відео-контентом.

- Дослідження системи, яка реалізує алгоритм підтримки сервісу колективно-конкурентного планування створення відео-контентом.

- Веб-реалізація системи підтримки сервісу колективно-конкурентного планування створення відео-контенту.

Практичне значення одержаних результатів. Додаток є потужною системою, яка автоматизує процеси планування та монетизації «реакційного» контенту. Завдяки впровадженню алгоритму диференційованих за ціною черг, система вирішує проблему, коли під час інтерактивних трансляцій, де велика кількість глядачів, відбувається хаотичне замовлення контенту, що робить проведення тих заходів творцям, практично, неможливо. Розроблений веб-сервіс дозволяє контент-мейкерам відмовитись від незручних ручних методів ведення списків замовлених відео, що значно полегшує життя стрімера під час ефіру. Автоматичний розрахунок тривалості перегляду на основі внесених грошей та обраного тарифу черги гарантує високу точність фінансових операцій та прозорість процесу для всієї аудиторії. Таким чином, результати роботи забезпечують перехід від хаотичного отримання запитів до керованої моделі формування ефірного часу.

Так чином система алгоритму підтримки сервісу колективно-конкурентного планування створення відео-контенту, є актуальною задачею для вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти, впровадження якої дозволяє трансформувати пасивний перегляд відео у динамічний процес з високим ступенем утримання уваги користувачів.

К6ПЗ_2026

Вим	Дру	№ докум	Підпис	Дата

ВКРБ-122.26.0023.00.00.ПЗ

Арк.

5

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Система розробляється як веб-сервіс для автоматизації колективно-конкурентного планування відео-контенту в реальному часі. Основне призначення – надання авторам цифрового контенту моделі, яка дозволяє аудиторії безпосередньо впливати на черговість відтворення контенту шляхом фінансової підтримки автора.

А також:

- Усунення “анархії” з чергами, запобігаючи “хаосу”, під час спільного перегляду контенту, коли користувачі можуть постійно переривати один одного та створювати велику кількість запитів одночасно, яких важко впорядкувати та пріоритезувати.

- Монетизація моделі “Priority Queuing”, що первісно є одною з абстрактних типів даних в комп’ютерних науках, де кожен елемент пов’язаний з деяким пріоритетом і чим він вище, тим найпершим він обслуговується.

- Гнучкість для контенту ефіру – творцю не потрібно знаходити попередньо контент для стріму, бо його пропонують глядачі в реальному часу.

- Створення зручного інтерфейсу користувача – більшість технічних рішень мають не user-friendly та не інтуїтивний інтерфейс користувача.

Можливості системи:

- Динамічне створення необмеженої кількості черг з будь-яким ім’ям, встановлення індивідуальної вартості для кожної черги, видалення черг за бажанням та динамічна зміна імені та ціни черги.

- Система забезпечує глядачу перегляд контент-творців, на яких він підписався, та переглядати їх список замовлених відео, а також самостійно замовляти відео-контент з платформи YouTube, обирати доступні черги, в залежності від бажаного пріоритету.

Вим	Дир	№ докум	Підпис	Дата

ВКРБ-122.26.0023.00.00.ПЗ

Арк.

6

- Часткове замовлення відео – якщо користувач не може замовити відео повністю він може замовити тільки частину відео, довжина якої залежить від черги та ціни за чергу, а інша частина аудиторії може до замовити решту відео.
- Фільтрація черг – глядачі та творці можуть фільтрувати загальний список замовлень за чергою для кращого орієнтування в списку замовлених відео.
- Творцю стріму відображається список підписників, людей які підписані на його список черг та відео.
- Система забезпечує миттєве та динамічне відображення списку замовлених відео без потреби оновлення сторінки.

1.2 Область застосування

Система підтримки сервісу колективно-конкурентного планування створення відео-контенту охоплює широку сферу сучасних онлайн-трансляцій та платформ, які базуються на інтерактивній взаємодії між контент-творцями та аудиторією. Ключовою областю впровадження розробки є сектор ОТТ-технологій, де спостерігається стрімке зростання попиту на персоналізовану доставку відео через мережу Інтернет. Додаток орієнтований на платформу YouTube, яка не тільки в Україні, а й усьому світі є абсолютним лідером серед систем розповсюдження відео-контенту. Використання моделей гейміфікації у процесі планування дозволяє значно підвищити залученість глядачів, що відповідає глобальним трендам розвитку сучасної медіа-індустрії. Сервіс виступає універсальним інструментом для впорядкування колективних запитів користувачів, забезпечуючи високу швидкість реакції системи в реальному часі.

Сервіс ідеально підходить для використання стрімерами та блогерами та іншим користувачам, які спеціалізуються на створенні так званого “реакційного контенту”. У цьому жанрі автор прямого ефіру переглядає, аналізує та коментує медіа-контент, який надає йому його аудиторія. В умовах високого попиту на

Вим.	Дир.	№ докум.	Підпис	Дата

аналіз подій пов'язаних з війною в Україні, система може ефективно використовуватись блогерами та стрімерами для розподілення медіа-контенту в залежності від пріоритетності новин або сюжетів, або відео-контенту на тему, які потребують розбору за запитом спільноти.

Користувачам, які хочуть пріоритезувати відео-контент з платформи YouTube в залежності від власних потреб. Завдяки автоматизації розрахунку вартості замовлень, сервіс звільняє контент-мейкерів від необхідності використання незручних “ручних” систем планування на кшталт Google spreadsheets чи текстових документів. Такий підхід стимулює глядачів більш активно приймати участь в плануванні відео-контенту та конкурувати за пріоритет перегляду, що позитивно впливає на загальну динаміку та монетизацію трансляції.

Окрім професійної стрімінгової діяльності, веб-додаток чудово виступає як інструмент для спільного перегляду групою людей: друзі, родина та інші закриті групи людей, де ця група людей може впливати на порядок перегляду відео, за допомогою черг. Інноваційна механіка “до замовлення” дозволяє учасникам спільно фінансувати тривалість довгих роликів, що робить процес споживання контенту справді колективним та інтерактивним.

Отже, система підтримки сервісу колективно-конкурентного планування створення відео-контенту виступає універсальним інструментом для впорядкування колективних запитів користувачів на базі середовща YouTube. В тому числі, завдяки використанню сучасного стеку технологій та крос-платформності, користувачі мають можливість взаємодіяти з чергами з будь-яких пристроїв, включаючи персональні комп'ютери, ноутбуки та смартфони. Універсальність архітектури системи робить її потужним інструментом для візуалізації планів ефіру та дозволяє адаптувати її під індивідуальні потреби будь-яких користувачів.

Вим	Дну	№ докум	Підпис	Дата

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур та програмних рішень

Проведено огляд наступних існуючих систем, які підходять до теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти:

StreamElements – це веб-орієнтована платформа для стрімерів, набір хмарних інструментів для покращення трансляцій, які залучають аудиторію монетизувати контент творців. Платформою користуються 2,6 млн. користувачів. Також StreamElements має інструмент, який називається Media Request – це інструмент, який надає глядачам медіа-творця, використовуючи бали, ділитися медіа-файлами з YouTube, особливо музикою під час стріму.

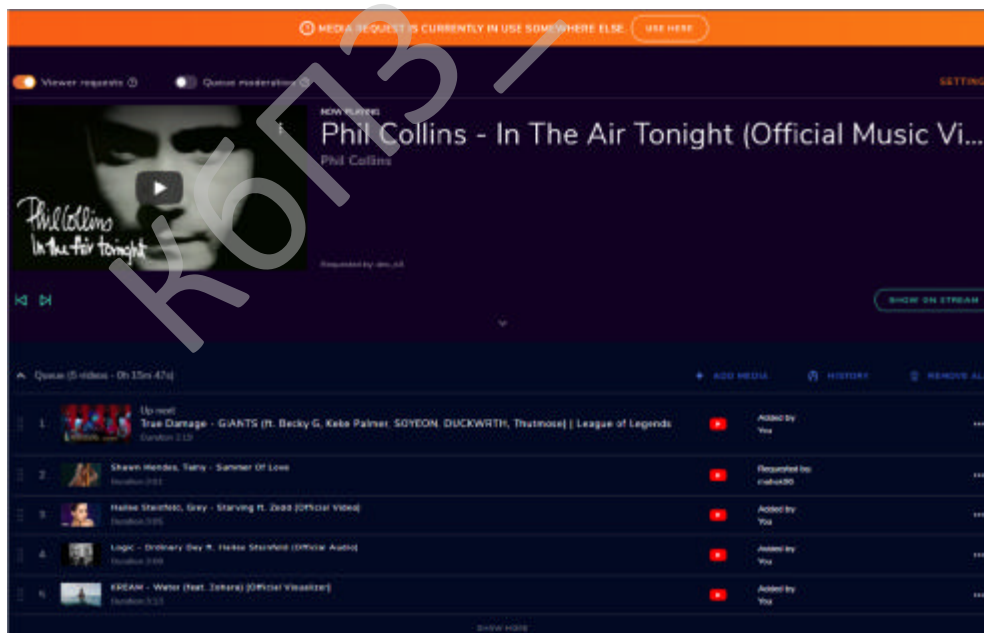


Рисунок 2.1 – Інтерфейс користувача StreamElements(Media Request)

Переваги:

- Платформа використовує хмарне середовище для своїх інструментів та зберігає налаштування та замовлення в хмарі.
- Користувачі можуть оплачувати свої замовлення, за допомогою донату або безкоштовно за “бали лояльності”. Також можна комбінувати обидва способи.
- Можна подивитись історію замовлених відео.
- Доступна фільтрація контенту: автоматичне блокування відео за тегами або за URL-адресами.
- Можна гнучко налаштовувати чергу замовлених відео (змінювати порядок відео, додавати та видаляти медіафайли), що можна побачити в інтерфейсі користувача на Рис. 2.1.

Недоліки:

- В системі є тільки одна черга, а значить відсутня пріоритезація за чергами.
- Неможливість колективно замовляти відео, користувач може замовити відео тільки повністю. Відсутня можливість замовити частково відео, а інший – може “докупити” додаткові хвилини. Це обмежує залученість деякої частини аудиторії.
- Стрімеру потрібно вручну перетягувати замовлення, щоб змінювати пріоритет відео в черзі.
- Додаток більше орієнтований на музичні запити.

Streamlabs Desktop – це безкоштовне програмне забезпечення для трансляції відео-контенту на велику кількість платформ з можливістю налаштування сцени, роздільної здатності та введення прямих трансляцій разом з іншими людьми. У цього програмного рішення є велика кількість віджетів, які можна додавати до сцени потокової передачі. Одним і з таких віджетів є Media Share.

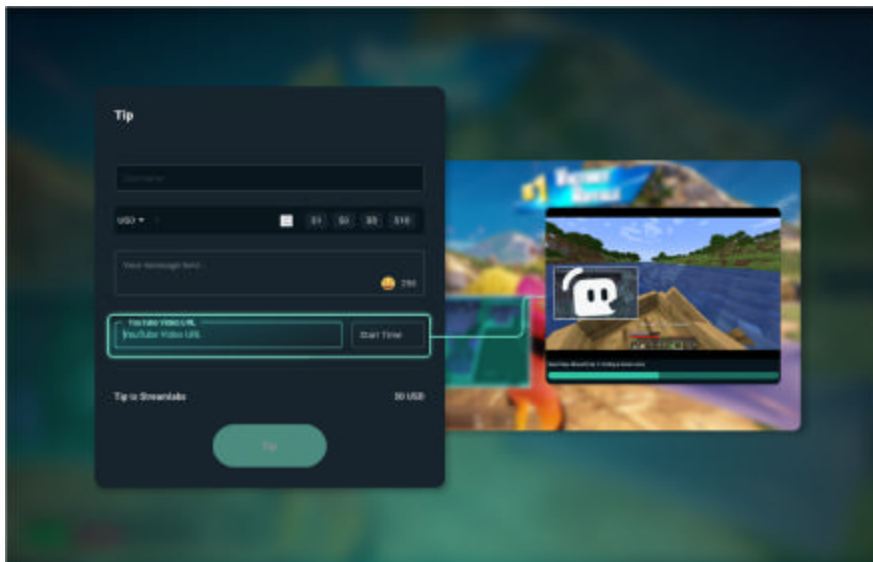


Рисунок 2.2 – Інтерфейс користувача для замовлення відео StreamLabs Desktop (Media Share)

Переваги:

- Програмне забезпечення є open-source.
- Користувачі можуть оплачувати свої замовлення, або безкоштовно через чат стріму або за допомогою донату.
- StreamLabs Desktop має величезну бібліотеку віджетів для гнучкого налаштування трансляції.
- Підключення віджетів в один клік.
- Доступна модерація контенту: до списку можна надати доступ модераторам стрімера, щоб вони фільтрували замовлені відео, також доступна фільтрація від спаму та вибір режиму відтворення відео: автоматично чи вручну.

Недоліки:

- Є тільки одна черга і її не можна налаштовувати.
- Як і в StreamElements відсутня можливість колективно-конкурентного замовлення відео.
- Не можна змінювати пріоритет в черзі.
- Користувачі критикують програмне забезпечення за високе навантаження на систему.

– В Excel-таблиці вже вбудовані деякі можливості, які можна використати при створенні такої системи – це формули та вирази, які можна використовувати для автоматичного розрахунку кількості замовлених хвилин.

– Конкретно в даній системі є фонди, які надають можливість глядачам донатити гроші на конкретні теми без конкретного відео, а потім аудиторія може замовити відео на тематику конкретного фонду без необхідності донатити повну суму, а використати суму з фонду.

– Також користувачі можуть створювати свої власні фонди, донатити туди гроші і використовувати їх потім.

– Є можливість “скіпнути” небажане відео глядачами, для цього потрібно задонатити в 2 рази більше грошей ніж вартість замовленого відео.

– Інтерфейс, доволі, інтуїтивний.

– Google Excel-таблиця знаходиться в хмарі. Нічого не зберігається на комп’ютері творця, що є захистом від втрати даних.

– Google Excel-таблиця оновлюється в реальному режимі.

Недоліки:

– Страшний інтерфейс користувача.

– Аудиторія залежить від сторонніх сервісів для донатів.

– Усі операції (додавання, видалення, підрахунок хвилин відео) робляться вручну.

– Усю систему і інтерфейс стрімеру потрібно створювати вручну.

Додатково існує, ще дві системи колективно-конкурентного планування створення відео-контенту, ще двох стрімерів, які також створені вручну: одна в Google Excel-таблиці, інша в Google Word-документі. Якихось визначних переваг чи недоліків в порівнянні зі системою стрімера Ubermarginal немає, окрім того, що система на основі Word-документу, має жахливий і не інтуїтивний інтерфейс та не має жодної автоматизації і, наче, взагалі покинута стрімером, через незручність.

Аналіз існуючих рішень у ніші колективно-конкурентного перегляду відео-контенту дозволяє зробити декілька висновків, що розробка алгоритму

дійсно потрібна і буде мати попит. Бо популярні платформи, такі як StreamElements та StreamLabs обмежені однією лінійною чергою і не мають декількох черг з диференційованою ціною за хвилину контенту, що сильно обмежує взаємодію контент-творця та його аудиторії, а також обмежує можливості монетизації для стрімерів “реакційного” контенту. Також аналіз “польових” даних демонструє, що серед стрімерів є попит на систему колективно-конкурентного планування створення відео-контенту але усі вони є ручними розробками з малою кількістю автоматизації і, як правило, жахливим та не інтуїтивним для глядача інтерфейсом, що призводить до покидання ніші творцями.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Для розробки алгоритму підтримки сервісу колективно-конкурентного планування створення відео-контенту потрібно обрати підходящий стек технологій, який забезпечить високу продуктивність в реальному часі, масштабованість розробки та надійність збереження даних.

Frontend. Для створення користувацького інтерфейсу обрано фреймворк React версії 19.2.7, мову програмування TypeScript та синтаксис препроцесору SCSS для каскадних стилів.

React (також відомий як: React.js, ReactJS) – це відкрита бібліотека для мови програмування **JavaScript** з можливістю використання **TypeScript**, який дозволяє створювати веб-застосунки, які використовують швидко змінювані дані і не хочуть через це постійно перезавантажувати сторінку. За допомогою React можна створювати інтерфейси користувачів з окремих частин, які називаються компоненти. Потім ці компоненти об’єднуються в цілі сторінки та додатки. React дозволяє створювати малі шматочки коду, а потім збирати їх як пазл, тому я віддаю перевагу саме цій бібліотеці.

Звичайно, існують альтернативи – це **Vue.js** та **Angular**, але я обрав React, через брак досвіду розробки на Vue та Angular. Також React дуже швидкий, легкий, масштабований та має велике ком'юніті розробників.

TypeScript – це строго типізована мова програмування, яка розроблена на JavaScript. Основні переваги TypeScript перед JavaScript – це надання додаткового синтаксису для виявлення помилок в редакторі ще на етапі написання коду, що прискорює розробку додатку, робить його надійнішим та передбачуваним. TypeScript чудово взаємодіє з React, бо типізує аргументи (props) для компонентів і тим самим, наприклад, зменшує кількість ситуацій коли потрібно пригадувати аргументи.

Додатково, використовується бібліотека **React Router** для маршрутизації в React-застосунків, полегшуючи створення кількох сторінок в односторінковій програмі та навігацію між ними, а також обробку параметрів в URL-адресах. Без цієї бібліотеки маршрутизація в React неможлива.

Для оптимізації додатку було обрано **Babel Plugin React Compiler**, який зменшує постійні повторні перемальовування компонентів, тобто прискорює роботу системи.

Sass (Syntactically Awesome Stylesheets) – скриптова метамова, яка інтерпретується в CSS та спрощує його, підвищуючи рівень абстракції.

SCSS – це один із самих популярних Sass-препроцесорів, який найближчий до CSS та збільшує його читабельність, за допомогою вкладеності коду та додавання змінних для збереження даних і використання їх в різних місцях одночасно.

Backend. Серверна частина реалізовується на платформі Node.JS, з використанням TypeScript.

Node.js – це безкоштовне, крос-платформне середовище для створення серверних частин веб-додатків, скриптів та інструментів командного рядка. За допомогою неї можна створювати додатки на JavaScript/TypeScript поза браузерами, за допомогою двигуна виконання JavaScript під назвою V8. Node.js проста та працює дуже швидко.

Так як Node.js має подійно-орієнтовану архітектуру та неблокуючу модель, які здатні виконувати асинхронні операції вводу/виводу, що робить її чудовим вибором для застосунків, які працюють та оновлюють дані в режимі реального часу, що якраз є основою сервісу колективно-конкурентного планування створення відео-контенту [29 – 30].

База даних. Для збереження даних обрано PostgreSQL, що є системою керування реляційними базами даних (СКБД). Проект з відкритим кодом написаний мовою програмування C, який використовують багато компаній по всьому світу. PostgreSQL забезпечує дотримання принципів ACID:

- Атомарність (Atomicity) – гарантує, що жодна транзакція не буде виконана тільки часткова. Будуть виконані або всі операції або жодна операція не буде виконана і транзакція буде відхилена.

- Узгодженість (Consistency) – система має бути узгодженою та несуперечливою, що будь-які дані, які записані в базу даних, обов’язково мають відповідати усім правилам та обмеженням.

- Ізольованість (Isolation) – гарантує інкапсуляцію змін всередині транзакції і робить їх невидимими і недоступними для інших транзакцій.

- Довговічність (Durability) – гарантує, що незалежно від проблем результат транзакції, які вже завершилися будуть збережені в базі даних.

Також, PostgreSQL забезпечує можливість переглядати дані в базі даних у режимі реального часу та дозволяє створювати зв’язки між таблицями, за допомогою зовнішніх ключів.

Для полегшення роботи з SQL, використовується **Prisma** – об’єктно-реляційний маппер (ORM), який використовується разом з базами даних, наприклад PostgreSQL. За допомогою Prisma створюються моделі, які є програмним представленням таблиць в базі даних в вигляді структур, що робить роботу з базами даним швидше та ефективніше.

Основним інструментом розробки обрано **Visual Studio Code**. Це “король” серед редакторів коду, бо:

- Безплатний.

- Крос-платформний.
- Має величезну кількість користувачів та розширень.
- Інтелектуальне автодоповнення коду (IntelliSense).
- Вбудовану роботу з Git
- Споживання невеликого рівня ресурсів.

Обраний стек технологій забезпечує оптимальний баланс між продуктивністю системи в реальному часі та швидкістю розробки, а також забезпечує надійне збереження даних.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускню кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає алгоритм підтримки сервісу колективно-конкурентного планування створення відео-контенту.

Під час розробки випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови алгоритму підтримку сервісу колективно-конкурентного планування створення відео-контенту. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатації;

е) сформулювати висновки про виконаний обсяг робіт та одержані результати.

Виконання усього зазначеного обсягу робіт дозволило трансформувати теоретичні засади алгоритму на цілісну та технологічно обґрунтовану систему підтримки колективно-конкурентного планування створення відео-контенту.

КБПЗ_2026

Вим	Дру	№ докум	Підпис	Дата

ВКРБ-122.26.0023.00.00.ПЗ

Арк.

18

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Розроблюваний алгоритм підтримки сервісу колективно-конкурентного планування створення відео-контенту призначений для автоматизації процесу замовлення медіа-контенту глядачами для творців шляхом монетизації ефірного часу, за допомоги системи диференційованих за ціною черг. В цій системі глядачі можуть впливати на контент потокової передачі як індивідуально, так і колективно.

Система побудована за принципом універсального доступу. Кожен користувач, який зареєструвався наділений можливістю переглядати контент та замовляти, а також бути творцем і створювати та налаштовувати систему черг та керувати замовленими відео.

Функціонування системи базується на трьох основних етапах життєвого циклу замовлення контенту:

- стан конфігурації: творець налаштовує параметри свої сторінки, створює список черг, з необмеженою кількістю самих черг, та встановлює тарифи для цих черг, як йому заманеться.

- стан активного планування: глядач знаходить або обирає стрімера через пошук або список підписок, обирає бажане відео на платформі YouTube та замовляє його, вносячи оплати за обрану кількість хвилин, враховуючи тарифи черг, яку користувач сам обрав.

- стан виконання: система автоматично розміщує замовлення в черзі згідно з обраним пріоритетом та відображає актуальний список замовлених відео у реальному часі, за допомогою механізму WebSocket.

WebSocket – це двонаправлений протокол зв'язку, за допомогою якого сервер та клієнт можуть надсилати один одному повідомлення. Кожна сторона одночасно відправляє та приймає повідомлення. Обмін даними відбувається по

Вим	Дир	№ докум	Підпис	Дата

єдиному каналу зв'язку, який відкритий протягом усієї сесії комунікації серверу та клієнта, при цьому в будь-який момент будь-яка зі сторін може його закрити.

Згідно специфікації WebSocket визначено дві URI схеми:

- ws:// для нешифрованого з'єднання.
- wss:// для шифрованого з'єднання.

Протокол складається з handshake та обміну даними.

Handshake – це процес встановлення з'єднання між сервером та клієнтом у WebSocket.

Процес “рукоштовування” відбувається за декілька кроків:

а) Запит на оновлення HTTP ініційований клієнтом, який вказує на намір перейти на протокол WebSocket.

б) Після отримання запиту сервер відповідає кодом стану HTTP 101, що вказує на успішний перехід на протокол.

в) Сервер та клієнт обмінюються заголовками, що містять інформацію для встановлення WebSocket з'єднання. Клієнт генерує випадковий ключ і кодує його за допомогою base64.

г) Сервер перевіряє отриманий ключ, створює свій ключ відповіді, кодує його і відправляє його клієнтові.

На рисунку 3.1 та 3.2 зображено вигляд handshake-запиту з боку клієнта та сервера.

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 8
```

Рисунок 3.1 – Запит від клієнта з наміром перейти на WebSocket протокол

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+x0o=
Sec-WebSocket-Protocol: chat
```

Рисунок 3.2 – Підтвердження переходу на протокол WebSocket від серверу

Конкурентним аспектом системи є можливість глядача обирати чергу з вищим пріоритетом, що гарантує йому більш швидше опрацювання його замовлення за інших.

Колективна складова реалізується через механіку “дозамовлення”: якщо глядач замовив лише частину відео (наприклад, тільки 5 хвилин з 15), то інший глядач або група глядачів можуть спільно доплатити за наступні хвилини, тим самим, продовжуючи час перегляду відео стрімером.

Авторизація користувача відбувається після успішної генерації JWT-токена на стороні сервера і передачі його клієнтові, після цього JWT-токен буде зберігатись на стороні клієнта.

JWT-токен – це стандарт токена доступу на основі JSON, який використовується для передачі даних для аутентифікації в клієнт-серверних архітектурах. Створюється токен на стороні сервера, де він підписується секретним ключем і передається на сторону клієнта.

JWT-токен складається з:

- а) Заголовок – json, в якому описаний тип токена та методи шифрування;
- б) Корисного навантаження – json, дані які передаються на клієнт;
- в) Підпис – створюється на основі перших двох елементів;

Після створення JWT-токен додається до кожного HTTP-запити в заголовку Authorization.

При додаванні замовлення в список замовлених відео на стороні клієнта відбувається запит через формат oEmbed та отримується дані про відео на платформі YouTube, яке глядач замовляє, а конкретно назва відео та прев'ю.

oEmbed – це API, який дозволяє веб-сайтам відображати вбудований контент (наприклад відео), та отримувати інформацію про цей об'єкт в вигляді JSON.

Стрімер має повний контроль за списком замовлених відео через свій Dashboard, де він може: змінювати назви черг, змінювати тарифи для черг, створювати та видаляти черги, видаляти відео. Усі ці операції взаємодії між сервером та клієнтом відбуваються за принципами REST API.

REST API – це архітектурний стиль набір правил для обміну даними через Інтернет з використанням HTTP методів [40 – 41].

Основні принципи REST:

- Stateless – режим “без стану”, сервер не повинен зберігати інформацію про попередні запити від клієнта. Більше надійності.
- Автономність клієнта та сервера – клієнт та сервер повинні розвиватись окремо один від одного.
- Універсальні інтерфейси та засоби, наприклад HTTP-методи або коди взаємодії.
- Кешування збільшує продуктивність програм.

Основні методи HTTP:

- GET – отримати дані.
- POST – створити дані.
- PUT – оновити дані або створити нові, якщо їх не існує.
- PATCH – частково оновити дані.
- DELETE – видалити дані.

В залежності від операції, яку треба виконати над даними, залежить вибір HTTP-методу.

Основні типи даних, які можна передавати між сервером та клієнтом:

- JSON – найпоширеніший формат структурованих даних.
- XML – застарілий формат, поступається JSON.
- Plain Text – звичайний текст.
- HTML – веб-сторінки або HTML-код.

- Binary Data – бінарні дані, такі як зображення або документи.
- Form Data – форматовані дані, дані з HTML-форм.

Основні статус коди:

- 101 – сервер дозволив перехід на інший протокол.
- 200 – запит успішно виконаний.
- 201 – успішне створення нових даних.
- 301 – веб-ресурс перемістився на нову постійну адресу.
- 400 – помилка на стороні клієнта.
- 401 – потрібна аутентифікація.
- 403 – відсутність прав доступу.
- 404 – сторінка не знайдена.
- 410 – ресурс був видалений назавжди.
- 429 – занадто багато запитів від клієнта.
- 500 – помилка на стороні сервера.
- 503 – сервер тимчасово недоступний.
- 505 – сервер не підтримує версію HTTP.

Передача даних в системі колективно-конкурентного планування відео-контенту відбувається через протокол HTTPS, який передає дані в зашифрованому вигляді. Що гарантується цифровим сертифікатом TLS. Це те що відрізняє HTTPS від звичайного HTTP.

TLS – криптографічний протокол для безпечної передачі інформації (електронна пошта, обмін файлами тощо) в мережі Інтернет. Є наступником SSL (Secure Sockets Layer). Використовує асиметричне шифрування.

Збереження складної структури взаємозв'язків між користувачами, підписками, ціновими чергами та замовленнями забезпечує СКБД PostgreSQL. Яка в поєднанні з Prisma ORM дозволяє підтримувати цілісність даних при конкурентних фінансових транзакціях.

Вим	Дов	№ докум	Підпис	Дата

3.2 Розробка структурної схеми

Структурна схема алгоритму підтримки сервісу колективно-конкурентного планування створення відео-контенту зображена на рисунку 3.3. Схема відображає загальну архітектуру системи та її основні компоненти, також основні взаємодії між серверними та клієнтськими частинами та мережевою інфраструктурою.

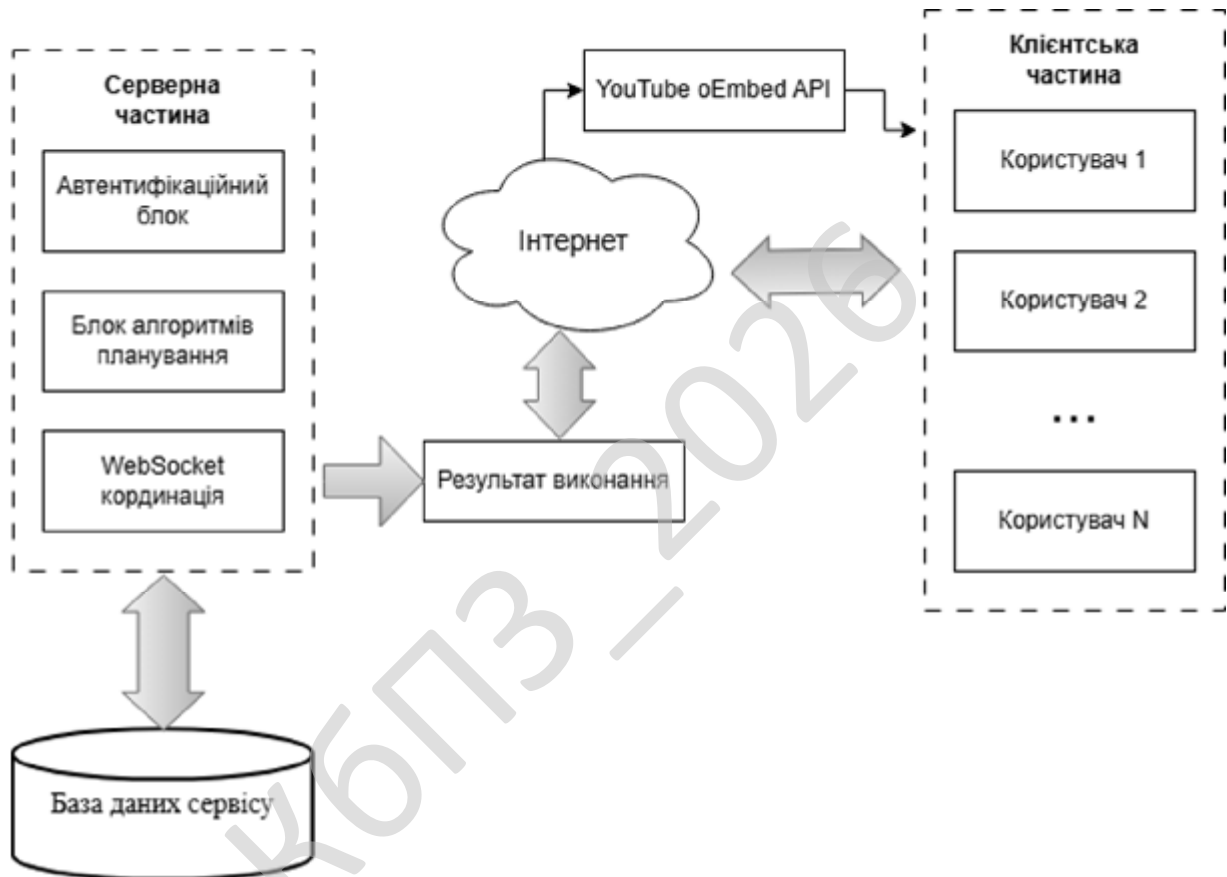


Рисунок 3.3 – Структурна схема системи

Система складається з трьох основних частин:

- серверна частина;
- база даних сервісу;
- клієнтської частини;

Серверна частина та клієнтська спілкуються за допомогою REST API архітектури, використовуючи Інтернет. Також додатково використовується протокол oEmbed YouTube API.

Клієнти з 1 по N – це кінцеві пристроїв стрімерів та глядачів, будь-які: персональні комп’ютери, ноутбуки, телефони. Через веб-браузер запускається інтерфейс системи побудований на базі бібліотеки React 19.2.7. Користувачі, через механізм перемикання ролей, можуть обрати сторінку “глядача” або сторінку “стрімера”.

На сторінці клієнта відбувається основна взаємодія користувача з системою. Користувач в режимі стрімера контролює систему черг (створює, змінює, видаляє) та диференціює їх за ціною, а також контролює замовлення його глядачів, маючи детально-необхідну інформацію про замовлений відео-контент. Також творцю доступна функція перегляду підписників (аудиторія) та фільтрації. В режимі глядача відбувається колективно-конкурентне планування контенту. Користувач знаходить стрімера, за допомогою системи пошуку, підписується на нього та відкриває його список замовлень. Глядач бачить список та вирішує замовити відео або до замовити вже замовлене відео, або просто отримує актуальну інформацію про відео-контент.

Веб-сервіс має центральний вузол системи, який реалізований з використанням середовища Node.js та складається:

- 3 блоку автентифікації: відповідає за валідацію логінів, реєстрацію користувачів та видачу JWT-токенів, які забезпечують безпечний доступ клієнту до серверного API.

- 3 блоку алгоритмів планування: ці алгоритми реалізують логіку сервісу – розрахунок вартості замовлень згідно з тарифами черг, обробка конкурентних запитів, логіка колективного розширення часу перегляду відео.

- 3 блоку WebSocket-координації: керує кожним Dashboard-ом та забезпечує миттєве відображення подій (замовлення або до замовлення відео, зміна черги та ціни) для усіх учасників, які підключені.

Дані, між частинами системи, транспортуються за допомогою мережі Інтернет. Дане середовище забезпечує двосторонню швидку та надійну комунікацію та доставку інформації між сервером та клієнтом, використовуючи

спеціальні протоколи передачі даних, сукупність цих протоколів називають стеком протоколів TCP/IP.

Якщо пояснювати просто, то набір протоколів TCP/IP нагадують роботу пошти. Користувач відправляє дані, в вигляді конвертів (пакетів) та відправляє отримувачу, а TCP, як кур'єр, передає ці конверти до адресата і перевіряє, що усі дані дійшли цілими та неушкодженими, а також повідомляє, що лист дійшов до отримувача. IP, в свою чергу, є чітко описаним ланцюжком “поштових відділень”, які потрібно пройти, щоб добратися до адресата.

Також не можна оминати дуже важливу частину структури сервісу – це база даних на основі СКБД PostgreSQL, використовується для перманентного збереження даних, а сама СКБД гарантує безпечне цілісне збереження даних в таблиці User, Queue, VideoOrder, Subscription та зв'язки між ними. За допомогою маппера Prisma ORM, відбувається асинхронна та безпечна передача даних між серверною частиною та базою даних, також ORM надає типізовану та просту структуру, яка відображає сутності БД та їх зв'язки, та спрощену і ефективну мову запитів, а також надійну систему міграцій.

Також веб-додаток використовує зовнішній ресурс (YouTube oEmbed): Система взаємодіє із зовнішнім API відеохостингу YouTube для автоматичного отримання метаданих, таких як: назва відео та прев'ю зображення, за вказаним користувачем URL, що спрощує процес наповнення черг відео-контентом.

Взаємодія починається з ініціації команди користувачем на клієнтській частині, наприклад створення замовлення. Далі запит через Інтернет разом з даними транспортується до серверної частини, де відбувається перевірка користувача та виконуються необхідні розрахунки та фіксація даних в базі даних. Після цього WebSocket координує, через встановлений з клієнтом handshake, появу нових даних в системі з усіма клієнтами, які підписані та підключені до Dashboard-у конкретного стрімера.

Вим.	Дир.	№ докум.	Підпис	Дата

3.3 Розробка функціональної схеми

Функціональна схема відображає взаємодію модулів сервісу колективно-конкурентного планування створення відео-контенту та описує процеси обробки даних. На рисунку 3.4 зображена функціональна схема системи.

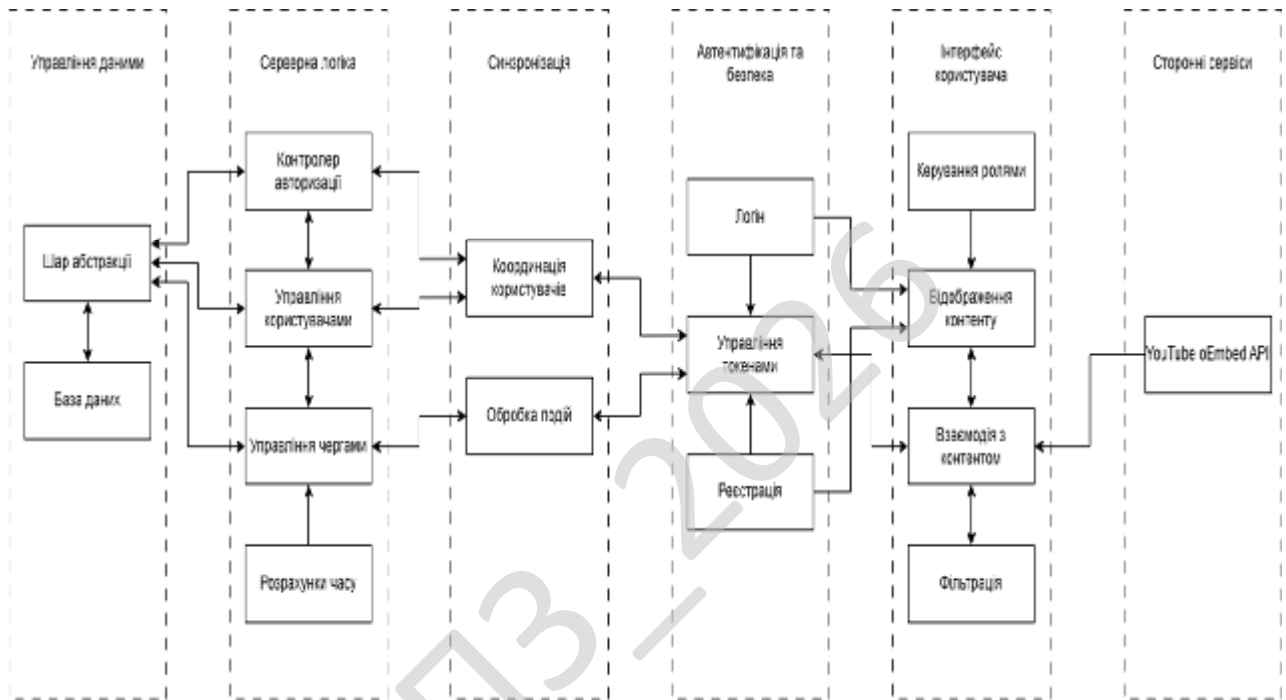


Рисунок 3.4 – Функціональна схема системи

Додаток поділяється на наступні функціональні модулі:

- Аутентифікація та безпека – керування логіном та реєстрацією через генерацію JWT-токенів.
- Інтерфейс користувача – візуалізація станів та процесів сервісу, а також взаємодія з кінцевим споживачем.
- Сторонні сервіси – зовнішній сервіс отримання даних про YouTube-відео.
- Синхронізація – забезпечення оновлення інтерфейсу в реальному часі без перезавантаження.

– Серверна логіка – модуль обробки черг, замовленого відео-контенту, колективно-конкурентних запитів.

– Управління даними – відповідає за збереження даних в базу даних.

Сервіс реалізує наступні функції всередині себе:

– Логін – це процес перевірки облікових даних. Сервер порівнює введену користувачем email-адресу та пароль з даними в базі даних та відповідає користувачеві про результат порівняння (успішна автентифікація або сталась помилка) та дає йому доступ до основних алгоритмів сервісу.

– Реєстрація – створює обліковий запис нового користувача, перевіряючи унікальність імені та електронної пошти користувача. У разі успіху передає дані користувача на сервер для збереження в базі даних та надає доступ до основної сторінки.

– Управління токенами – для доступу до функції додатку, після логіну або реєстрації, користувачеві надається JWT-токен, який зберігається на стороні клієнта та додається до кожного запиту користувача в заголовок HTTP-запиту.

– Керування ролями – це перемикач “I am viewer / I am streamer”, який динамічно змінює інтерфейс та набір доступних функцій на сторінці.

– Відображення контенту – клієнтський модуль, який відповідає за графічну візуалізацію можливостей сервісу та доступного користувачеві контенту.

– Взаємодія з контентом – модуль, який надає клієнту можливість впливати на контент, наприклад, змінювати черги (їх ім’я та ціну), видаляти відео, шукати нових стрімерів, замовляти нові відео на конкретній сторінці творця, також виконує запит до стороннього сервісу YouTube oEmbed.

– YouTube oEmbed API – сторонній сервіс для автоматичного отримання метаданих відео, таких як назва та прев’ю, за заданим URL-посиланням на відео з платформи YouTube.

– Фільтрація – забезпечує фільтрацію замовлень за чергами, надаючи користувачам можливість бачити тільки ту інформацію, яку вони потребують та відсіювати усі зайві відео.

– Координація користувачів – автоматично надає користувачеві актуальну інформацію про черги, їх ціну та замовлені відео конкретного стрімера, а також актуальну інформацію про підписників.

– Обробка подій – фіксує усі зміни на клієнтській частині додатку, які виконують користувачі (зміна Dashboard, видалення відео, зміна черг та ціноутворення, колективне до замовлення відео) та в реальному часі, за допомогою технології WebSocket, оновлює інформацію в базі даних та надає цю оновлену інформацію тим користувачам, які пов'язані з подією.

– Контролер авторизації – генерує JWT-токени та забезпечує безпеку застосунку та даних користувачів.

– Управління користувачами – обробляє усі процеси логіну та реєстрації користувачів (перевірка, збереження даних в БД), оперує зміною ролей (глядач / стрімер) та пошуком стрімерів за ім'ям.

– Управління чергами – усі основні дії користувачів, які надаються сервісом колективно-конкурентного планування створення відео-контенту, обробляються в даному модулі. Усі дані про черги та відео, які були створені, змінені або видалені, зберігаються та актуалізуються в базі даних тут. Також передає дані, які повинні бути відображені на клієнті.

– Розрахунок часу – виконує розрахунок кількості хвилин перегляду на основі суми, яку внесли на відео глядачі, та встановленого стрімером тарифу.

– Шар абстракції – забезпечує швидкий та типізований доступ до таблиць бази даних та даних в них, надаючи легкий інструментарій для запитів та зрозумілі моделі таблиць в вигляді структур.

– База даних – зберігає моделі глядачів, стрімерів, черг та відео, а також усі зв'язки між ними, зберігає усі дані, на постійній основі, в захищеному та надійному місці.

3.4 Розробка діаграми процесів

На рисунку 3.5 зображена діаграма процесів алгоритму сервісу колективно-конкурентного планування відео-контенту, яка демонструє життєвий цикл взаємодії користувача з системою, також демонструє дії та переходи, які відбуваються під час взаємодії.



Рисунок 3.5 – Діаграма процесів сервісу.

Функціонування сервісу можна поділити на декілька етапів та циклів. Логіка роботи системи побудована за подієво-орієнтованою моделлю, де кожна дія ініціює ланцюжок перетворень від клієнтського запиту до актуалізації стану в базі даних та розсилки оновлень учасникам:

а) Етап автентифікації та ініціалізації:

Початок – Автентифікація користувача – Обробка операцій на сервері – Збереження в базі даних – Dashboard

Після відкриття веб-застосунку, система переходить до автентифікації та безпеки, після введення облікових даних, сервер виконує перевірку через базу даних логіну або реєстрації, генерує унікальний JWT-токен, який передається користувачу та фіксується в браузері клієнта. Після чого користувач потрапляє до Dashboard.

б) Етап вибору ролі:

Центральним елементом Dashboard є блок керування ролями, який дозволяє користувачу миттєво перемикатися між статусами «стрімер» та «глядач». Кожен зареєстрований учасник системи має універсальний доступ і може виступати в обох ролях без створення окремих акаунтів.

Режим стрімера – надає доступ до інструментів конфігурації черг та контролю замовлень.

Режим глядача – передбачає функції пошуку стрімера, підписки на нього, вибору творця і відписки від нього, за бажанням.

в) Цикл фільтрації:

Для зручного орієнтування у великих обсягах інформації в системі передбачено спеціалізований модуль фільтрації. Цей цикл надає можливість як стрімеру, так і глядачу сортувати контент за конкретними чергами, щоб бачити лише пріоритетні замовлення.

г) Процеси в режимі стрімера:

Творець керує сервісом через чотири процеси: додавання, редагування та видалення черги, також видалення відео. Стрімер має повний контроль над ціноутворенням, що дозволяє йому гнучко адаптувати монетизацію під поточну аудиторію.

г) Процеси в режимі глядача:

Глядач може знайти або обрати стрімера і тим самим, підписатись або відписатись від творця-контенту. Також глядач може ініціювати запит на

Вим.	Дир.	№ докум.	Підпис	Дата

замовлення відео, що, в свою чергу, ініціює запит до YouTube oEmbed API для отримання метаданих. Користувачі колективно мають змогу до замовити відео, вносячи додаткові гроші.

д) Процес обробка операцій на стороні сервері та відображення на Dashboard:

Обробка операцій на сервері – Збереження в базі даних – Отримання результату – Dashboard

Кожен процес, від створення замовлення до зміни ціни, супроводжується обробкою на серверній стороні сервісу та збереженням в базі даних нових даних. Після цього, за допомогою WebSocket отримується актуальні дані в режимі реального часу та відправляються на Dashboard. Це дозволяє всім підключеним учасникам отримати актуальні дані без примусового перезавантаження сторінок браузера, що забезпечує «живе» оновлення інтерфейсу.

е) Завершальний етап:

Dashboard – Кінець

Робота з сервісом колективно-конкурентного планування створення відео-контенту завершується виконанням процедури безпечного виходу, після виходу користувача з системи. На цьому етапі клієнтський додаток закриває WebSocket-сесії, одночасно з цим відбувається анулюванням активного JWT-токена шляхом його видалення з сховища браузера. Глобальний стан системи скидається до початкових значень, а користувач перенаправляється на Landing-сторінку. Така послідовність дій гарантує цілісність даних та захист персональної інформації клієнта.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ ТА ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ І ПРОГРАМНИХ РІШЕНЬ

4.1 Розробка блок-схем та опис алгоритмів функціонування системи

На рисунку 4.1 наведено блок-схему основної програми.

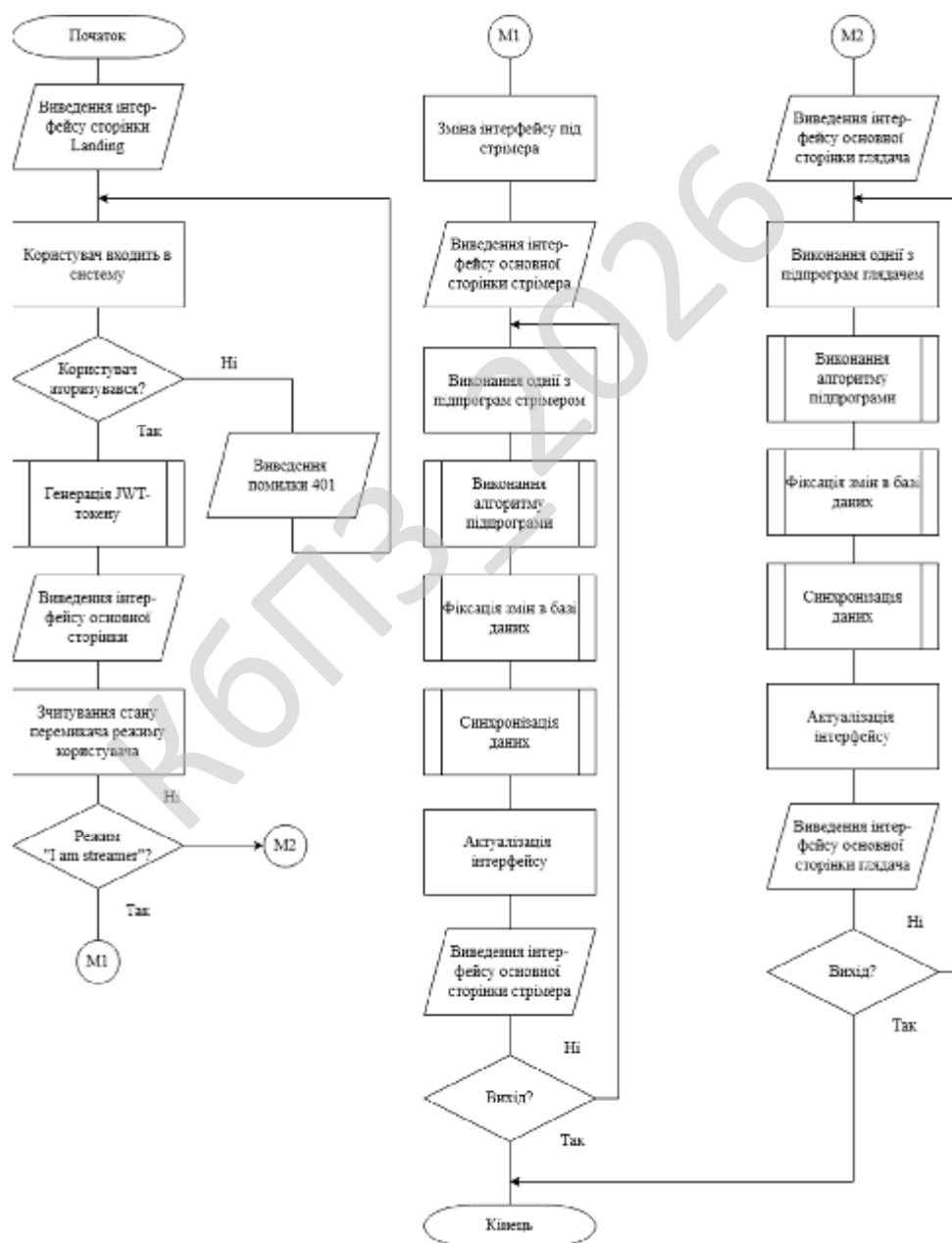


Рисунок 4.1 – Блок-схема роботи основної програми.

Алгоритм функціонування основної програми сервісу розроблено як інтегровану подієво-орієнтовану систему, що забезпечує життєвий цикл взаємодії користувача від моменту входу до синхронізації даних в реальному часі. На блок-схемі зображено розгалуження логіки залежно від обраної ролі користувача та стан збереження інформації. Програмно реалізація алгоритму базується на використанні середовища Node.js для бекенду та фреймворку React для фронтенду, що дозволяє досягти високої швидкодії та стійкості до конкурентних запитів. Кожен блок схеми відповідає конкретному функціональному модулю програми.

Процес ініціалізації системи починається з кроку запуску серверного модуля та ініціалізації глобальних налаштувань. На цьому етапі відбувається зчитування змінних середовища з файлу .env, зокрема рядка підключення до бази даних DATABASE_URL та секретного ключа JWT_SECRET для підпису токенів. Паралельно ініціюється підключення до СКБД PostgreSQL через Prisma та підготовка HTTP-сервера до прослуховування портів. Тільки після успішної перевірки готовності всіх ресурсів програма переходить до стадії взаємодії з користувачем.

Першим етапом взаємодії з користувачем є візуальна демонстрація інтерфейсу сторінки Landing, яка є кореневим вузлом додатку. Користувач бачить презентаційну сторінку, що містить заклики до дії (Get Started) та кнопки для авторизації, що відображаються в компоненті Header. Алгоритм на фронтенді використовує хук useLocation() для автоматичного визначення типу заголовку. Сторінка Landing виконує функцію маркетингового шлюзу, після якого починається безпосередня робота з алгоритмами планування черг.

Хук useLocation() – це вбудована функція в бібліотеку react-router, яка повертає поточну URL-адресу на якій знаходиться користувач. Дуже корисно, наприклад, для зміни компонента програми, який використовується одночасно в різних місцях інтерфейсу, в залежності від поточної сторінки, на якій знаходиться користувач, сервіс демонструє один і той самий компонент, але з різним наповненням та функціоналом, що і використовується в програмі. Хук

нічого не приймає на вхід, а на виході повертає об'єкт `location`, який містить в собі наступні властивості:

- `pathname` – URL-рядок, який починається з символу `“/”`;
- `state` – унікальний рядок, який асоціюється з поточною локацією;
- `search` – URL-рядок, який починається з символу `“?”`;
- `key` – унікальний рядок, пов'язаний з цим розташуванням;
- `hash` – URL-рядок, який починається з символу `“#”`.

Наступним кроком відбувається процес збору та передачі облікових даних через компоненти `LoginPage` та `LoginForm`. Стрімер або глядач вказує свою електронну пошту та пароль, які проходять певну валідацію на стороні клієнта, і при помилці повідомляє користувачу. Після натискання кнопки входу клієнт формує POST-запит до серверу, передаючи зашифровані дані. Цей крок є критичним для безпеки, оскільки він ініціює ланцюжок автентифікації.

Далі система виконує порівняння введеного пароля з паролем, що зберігається в базі даних PostgreSQL. Якщо валідація не проходить, алгоритм повертає користувачеві помилку 401 з статусом `Unauthorized`. В інтерфейсі це відображається через текстові індикатори червоного кольору в універсальному компоненті `Input`, а доступ до сервісу залишається заблокованим. Користувачеві пропонується виправити помилку та пройти автентифікацію заново.

При успішному підтвердженні особи алгоритм виконує генерує JWT-токен, що забезпечує реалізацію `stateless` архітектури. Серверний контролер авторизації підписує унікальний рядок, який містить ідентифікатор користувача та термін дії токена, і повертає його клієнту в форматі JSON. Фронтенд-додаток отримує цей токен і надійно фіксує його в `localStorage`, одночасно оновлюючи глобальний `AuthContext` для автоматичного додання заголовка `Authorization: Bearer <token>` до всіх майбутніх запитів. Це дозволяє системі розпізнавати користувача під час роботи з сервісом.

Далі користувач потрапляє на центральний `Dashboard`. За замовчуванням система завантажує режим глядача, активуючи `DashboardLayout` з трьох-колонковою структурою (фільтри, контент, підписки). Клієнтський додаток

виконує початкове отримання даних з серверної частини через REST API, отримуючи список доступних стрімерів або власні підписки. На екрані з'являється `PickStreamerPage`, яка слугує стартовою точкою для процесу колективного планування контенту.

Зчитування стану перемикача режиму відповідає за динамічну зміну профілю користувача через компонент `ModeToggle`. Алгоритм постійно відстежує стан змінної `mode` (змінна `union type` – тип, який може приймати лише певні значення, в програмі це рядки: `streamer` та `viewer`) у глобальному контексті `DashboardContext`, що дозволяє миттєво реагувати на дії користувача. При кожному перемиканні відбувається очищення поточного стану черг та автоматична навігація за адресою `/dashboard`, щоб запобігти конфліктам даних між різними ролями. Цей механізм забезпечує гнучкість системи, де кожен зареєстрований аккаунт одночасно має функції і стрімера, і глядача.

Подальший шлях виконання програми ділиться на два напрямки в залежності від обраного режиму. Якщо користувач обрав роль стрімера, система ініціює перемикання прапорця `isOwner` у положення `true`, що розширює права доступу до керування чергами. У гілці глядача основний акцент робиться на взаємодію з `OrderModal` та механіку `Extend`. Це розгалуження є фундаментальним для розмежування функціоналу в межах єдиного інтерфейсу застосунку.

Переходячи за з'єднувачем `M1`, система виконує зміна інтерфейс під стрімера, де компоненти сторінки адаптуються для потреб модерації. Правий частина сторінки `SubscriptionsSidebar` автоматично перемикається в режим відображення підписників через запит до серверу. Глядацькі кнопки замовлення відео замінюються на інструменти створення нових цінових рівнів через компонент `AddQueueForm`. Стрімер отримує повний візуальний контроль над власною сторінкою, що є необхідною умовою для керування трансляцією.

Активація однієї з підпрограм описує функцій модерації: створення черги, зміна ціни за хвилину або видалення відео. Кожна така дія ініціює виклик специфічного методу в шарі API, наприклад метод `PATCH` для актуалізації

тарифів. Стрімер взаємодіє з формами редагування в секції QueueSection, що дозволяє змінювати параметри ефіру без переривання трансляції. Система очікує підтвердження дії від користувача перед переходом до серверної обробки.

Якщо клієнтська частина ініціює виконання одного з алгоритмів сервер перевіряє права на виконання, використовуючи дані з JWT-токена. Якщо стрімер намагається видалити відео, алгоритм перевіряє, чи збігається його ID з полем streamerId у базі даних. У разі створення нової черги, система валідує назву та ціну, забезпечуючи коректність фінансових параметрів. Цей етап гарантує, що жодна критична зміна не буде внесена несанкціоновано або з помилковими даними.

Після успішного виконання підпрограми, за допомогою асинхронних операції Prisma ORM, дані зберігаються в базу даних. Зміни записуються в таблиці Queue або VideoOrder, причому операції видалення черг виконуються з правилом CASCADE, що автоматично прибирає всі пов'язані замовлення відео. База даних гарантує принципи ACID, що особливо важливо при зміні пріоритетів та цін у конкурентній черзі. Після успішного запису сервер отримує підтвердження і готовий до трансляції оновлень.

Ключовим є синхронізація даних на клієнті, щоб реалізувати real-time ефект через протокол WebSocket. Після кожної успішної транзакції в БД серверний емітер надсилає подію, наприклад queue:updated або order:removed, у кімнату списку замовлених відео стрімера. Це забезпечує миттєве сповіщення всіх підключених учасників, включаючи глядачів. Використання бібліотеки Socket.io дозволяє звести затримку до мінімуму, що критично для інтерактивного планування.

Далі клієнтський застосунок обробляє отриману WebSocket-подію. Компонент StreamerPage реагує на вхідний запит та оновлює локальний стан черг або списку відео без повного перезавантаження сторінки. Користувач бачить, як картка відео зникає або змінюється її позиція в черзі миттєво після

дії стрімера. Це створює відчуття безперервного потоку даних та прозорості процесу модерації для всієї аудиторії.

Інша гілка (Режим глядача) логіка починається виконання однієї з підпрограм глядачем, де основною дією є замовлення нового контенту. Глядач відкриває модальне вікно `OrderModal`, де вводить посилання на YouTube та обирає бажану цінову чергу. Система автоматично підказує вартість хвилини. Також доступна підпрограма `Extend`, яка дозволяє додати кошти до вже існуючого замовлення через форму `ExtendForm`.

Алгоритм підпрограми глядача включає запит метаданих через YouTube `oEmbed API`, що дозволяє автоматично отримати назву та прев'ю відео за 800мс після введення URL. Програма розраховує кількість оплачених хвилин як частку від суми донату до тарифу обраної черги. Це забезпечує точність фінансових розрахунків та позбавляє глядача необхідності вручну вводити дані про ролик. Тільки після повної валідації суми та URL замовлення відправляється на сервер.

Зміни фіксуються в БД створенням запису в таблиці `VideoOrder` із прив'язкою до `viewerId`. Система фіксує чергу замовленого відео для подальшого сортування за системою FIFO всередині однієї черги. Завдяки використанню PostgreSQL, дані глядачів надійно захищені та доступні для швидкого пошуку. Кожне нове замовлення стає частиною черги стрімера.

У обох гілках алгоритму перевіряє активність сесії та бажання користувача завершити роботу. Якщо користувач натискає кнопку виходу в меню аватара, ініціюється процедура `logout`, яка очищує `localStorage` від JWT-токена. Це рішення перериває цикл очікування нових команд та переводить програму до фінального стану. Безпечне завершення сесії запобігає несанкціонованому використанню токена.

Завершується робота алгоритму розірвання активного з'єднання з WebSocket-сервером. Програма припиняє прослуховування подій у кімнаті стрімера та звільняє ресурси пам'яті, що займали глобальні контексти. Всі фонові процеси клієнтської частини коректно зупиняються, а користувач

Вим	Дир	№ докум	Підпис	Дата

повертається на сторінку Landing. Це гарантує цілісність сесії та завершує повний функціональний цикл сервісу.

На рисунку 4.2 наведено блок-схему підпрограми замовлення відео глядачем.

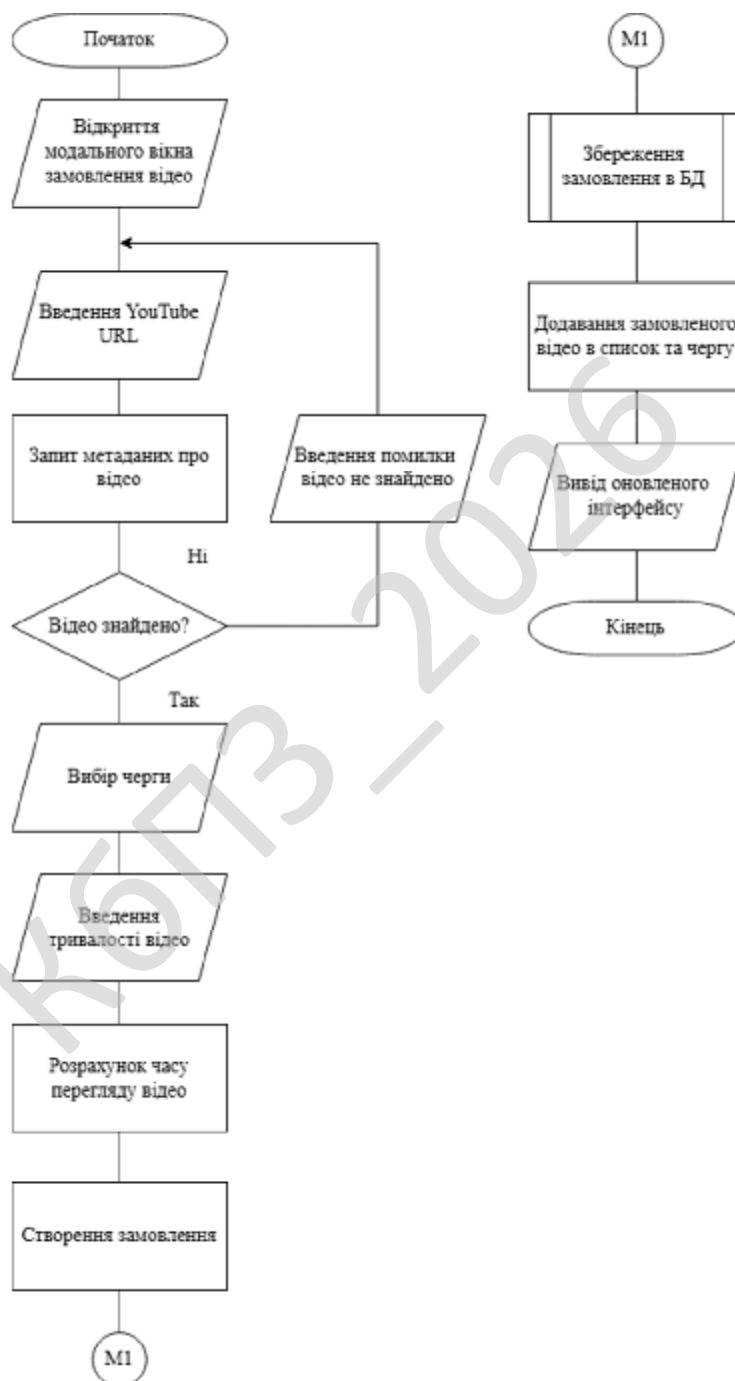


Рисунок 4.2 – Блок-схема підпрограми замовлення відео глядачем.

Алгоритм підпрограми замовлення відео глядачем є критично важливим компонентом сервісу, оскільки саме він реалізує колективно-конкурентну механіку взаємодії аудиторії з контентом. Алгоритм ініціюється в момент натискання користувачем кнопки “Order video” у верхній навігаційній панелі Dashboard. Ця дія активує стан відкриття спеціалізованого компонента OrderModal, який слугує інтерактивним інтерфейсом для збору необхідних параметрів майбутнього замовлення. Використання модального вікна дозволяє глядачу залишатися в контексті поточної сторінки стрімера, забезпечуючи зручність та безперервність користувацького досвіду.

Першим кроком активної взаємодії в межах відкритого вікна є введення URL-посилання на відео з платформи YouTube у відповідне текстове поле. Система використовує механізм відкладеної обробки тривалістю 800 мс, щоб почати аналіз посилання лише після того, як користувач завершить введення. Одразу після цього алгоритм автоматично ініціює запит метаданих через хук useYouTubeMeta, звертаючись до зовнішнього сервісу YouTube oEmbed API. Ця операція дозволяє системі отримати назву ролика та його прев'ю-зображення в форматі JSON без необхідності використання приватних API-ключів розробника на фронтенді.

Далі система аналізує отриману відповідь від сервера YouTube. Якщо введений URL є невірним або відео має обмеження за доступом, алгоритм переходить до гілки обробки помилок, виводячи повідомлення “Відео не знайдено або недоступне”. У такому разі користувачу пропонується скоригувати введені дані, а стан модального вікна повертається до початкової точки введення URL. Якщо ж метадані отримані успішно, інтерфейс динамічно оновлюється, відображаючи назву відео та його обкладинку для візуального підтвердження замовлення.

Наступним етапом є вибір цінової черги, де глядач визначає бажаний рівень пріоритету для свого замовлення. Алгоритм надає список доступних черг конкретного стрімера, кожна з яких має встановлений тариф. Глядач вводить суму фінансової підтримки в доларах, що супроводжується миттєвим

розрахунком часу перегляду за формулою відношення суми до тарифу черги. У випадках, коли oEmbed не повертає точну тривалість ролика, програма надає користувачу можливість ручного введення таймінгу у форматі MM:SS, що гарантує гнучкість системи.

Після фіналізації всіх полів відбувається безпосереднє створення об'єкта замовлення та перехід до збереження даних. Система формує POST-запит до серверу, передаючи ідентифікатори глядача, обраної черги та розраховані параметри часу. Серверний модуль, використовуючи Prisma ORM, асинхронно записує новий рядок у таблицю VideoOrder бази даних PostgreSQL. Такий підхід забезпечує цілісність фінансової операції та неможливість втрати даних при високій інтенсивності замовлень від різних глядачів одночасно.

Завершальним етапом роботи підпрограми є трансляція оновленого стану всім учасникам системи через механізм WebSockets. Сервер емітує подію order:created, що змушує клієнтські React-додатки всіх глядачів негайно додати нову картку відео до списку відтворення. Це забезпечує ефект “живої черги”, де глядач бачить результат свого внеску миттєво, без необхідності оновлення сторінки браузера. Після успішної синхронізації модальне вікно автоматично закривається, і підпрограма припиняє своє виконання, повертаючи користувача до основного циклу Dashboard.

Підпрограма роботи з YouTube oEmbed API. Підпрограма, реалізована у вигляді спеціалізованого React-хука useYouTube, є одним із ключових технічних вузлів фронтенд-частини системи, оскільки вона відповідає за автоматизоване вилучення метаданих відео-контенту. Основне функціональне призначення цього модуля полягає у забезпеченні безшовного зв'язку між клієнтським інтерфейсом та зовнішнім YouTube oEmbed API для отримання актуальної інформації про ролик за його посиланням. Використання такої архітектурної одиниці дозволяє значно спростити логіку компонентів замовлення відео (наприклад, OrderModal), виносячи складні асинхронні операції та керування станами в окрему ізольовану сутність. Програмна реалізація хука базується на принципах реактивного програмування із

використанням вбудованого засобу `useState` для відстеження життєвого циклу запиту. Завдяки цьому підходу система отримує можливість динамічно реагувати на введення користувача, автоматизуючи рутинні процеси заповнення заголовків та обкладинок відео, що безпосередньо впливає на якість користувацького досвіду та швидкість планування черг.

Стосовно інтерфейсу взаємодії, підпрограма `useYouTube` не приймає вхідних аргументів під час ініціалізації, але надає розробнику строго типізований об'єкт керування через інтерфейс `UseYouTubeReturn`. Основним вхідним параметром для активної роботи є рядок `url`, який передається у внутрішню функцію `fetchMeta` безпосередньо в момент завершення введення посилання глядачем. На виході підпрограма повертає комплексний стан, що складається з об'єкта метаданих `meta` та рядка помилки `error` у разі виникнення збоїв. Об'єкт метаданих структурований згідно з інтерфейсом `YouTube`, що включає назву відео (`title`), посилання на прев'ю-зображення (`thumbnail`) та розраховану тривалість у хвилинах (`durationMinutes`). Крім того, хук експортує функцію `reset`, яка дозволяє компоненту-споживачу миттєво очистити всі внутрішні стани підпрограми для підготовки до нового циклу замовлення.

Внутрішня логіка підпрограми починається з етапу валідації та обробки вхідного рядка за допомогою допоміжної функції `extractVideoId`, що базується на використанні регулярних виразів. Ця функція здатна розпізнавати різні формати посилань `YouTube`, включаючи стандартні адреси перегляду, скорочені лінки `youtu.be`. У випадку, якщо алгоритм не може знайти унікальний ідентифікатор відео, підпрограма негайно припиняє виконання та встановлює стан помилки "Невірне посилання `YouTube`", блокуючи подальші запити до зовнішніх мереж. Якщо ідентифікатор успішно вилучено, хук ініціює асинхронний перехід, переводячи систему в стан очікування через встановлення прапорця `loading` у положення `true` та очищення попередніх результатів. Такий підхід до попередньої обробки даних гарантує цілісність запитів та мінімізує навантаження на мережеві ресурси за рахунок відсікання некоректних вхідних даних ще на етапі клієнтської валідації.

Вим.	Дир.	№ докум.	Підпис	Дата

Центральним процесом підпрограми є виконання мережевого запиту до кінцевої точки `youtube.com/oembed`, яка повертає структуровані дані про відео у форматі JSON. Після отримання успішної відповіді алгоритм зчитує отриманий об'єкт для вилучення назви ролика, а за допомогою раніше отриманого ідентифікатора генерує пряме посилання на сервери зображень YouTube для формування обкладинки. Підпрограма також намагається отримати додаткові дані, такі як тривалість відео, через інтеграцію з проксі-серверами або сторонніми сервісами на кшталт `noembed.com`, що забезпечує максимальну інформативність замовлення. У разі виникнення виняткових ситуацій, наприклад відсутності доступу до відео або помилок сервера, блок `catch` перехоплює помилку та транслює її в інтерфейс користувача у зрозумілому вигляді. Завершується робота підпрограми в блоці `finally`, де відбувається скидання прапорця завантаження, що сигналізує головному циклу програми про готовність даних для подальшого розрахунку часу перегляду в черзі.

Підпрограма роботи до замовлення відео. Функція колективного до замовлення хвилин, реалізована через компонент `ExtendForm`, є технічним втіленням головної ідеї проєкту — спільного керування контентом. На відміну від стандартних систем, де замовлення є індивідуальним актом, цей модуль дозволяє аудиторії об'єднувати зусилля для продовження перегляду цікавих медіа-матеріалів. З погляду архітектури, підпрограма інтегрується безпосередньо в картку відео, що дозволяє глядачам взаємодіяти з чергою, не переходячи на інші сторінки та не втрачаючи фокусу на трансляції. Така механіка не лише посилює соціальну взаємодію всередині спільноти, а й виступає потужним інструментом монетизації, оскільки знімає з одного користувача фінансовий тягар повної оплати довготривалого ролика. Колективна природа цього алгоритму забезпечує динамічне наповнення ефіру контентом, який дійсно цікавий більшості присутніх глядачів.

Підпрограма отримує через інтерфейс критично важливі параметри: об'єкт поточної черги з її тарифом, кількість уже оплачених хвилин та загальну

тривалість відео. Внутрішній стан форми керується за допомогою хука `useState`, який відстежує введену користувачем суму в режимі реального часу. Важливою технічною деталлю є використання допоміжної функції `formatMinSec`, яка перетворює абстрактне числове значення типу `number` у зрозумілий для людини формат “хвилини: секунди”. Це дозволяє користувачу ще до моменту підтвердження оплати бачити, на скільки саме продовжиться ефірний час стрімера завдяки його внеску.

Логіка розрахунку додаткового часу базується на математичному відношенні внесеної суми до встановленого стрімером тарифу обраної черги. Алгоритм автоматично обчислює значення `additionalMinutes` як результат ділення `amountNum` на `queue.pricePerMinute`, забезпечуючи точність фінансових операцій до сотих часток. Паралельно система розраховує залишок вільного часу в ролику (`remainingMinutes`), що дозволяє візуально підказувати глядачеві межі можливого до замовлення. Для запобігання некоректним діям у компоненті реалізовано систему валідації: кнопка підтвердження залишається неактивною, доки користувач не введе валідне число. Такий підхід мінімізує помилки введення та гарантує, що на сервер будуть відправлені лише перевірені дані.

Процес підтвердження до замовлення ініціює асинхронний ланцюжок взаємодії з бекендом через метод `onExtend`. Клієнтська частина надсилає PATCH-запит на сервер, передаючи ідентифікатор відео та нову кількість хвилин. Серверний модуль, використовуючи Prisma ORM, виконує операцію оновлення поля `orderedMinutes` у базі даних PostgreSQL. Після успішного збереження в БД система переходить до етапу глобальної синхронізації.

Фінальний етап роботи підпрограми забезпечується технологією WebSockets, яка миттєво транслює подію `order:extend`. Це змушує інтерфейси всіх підключених глядачів автоматично оновити `progress bar` на відповідній картці відео. Глядачі бачать, як смужка оплаченого часу заповнюється в реальному часі, що створює ефект спільної участі у створенні трансляції. Після завершення події форма до замовлення автоматично закривається, повертаючи

користувача до режиму спостереження за чергою. Таким чином, функція Extend перетворює статичний список відтворення на “живий” організм, стан якого постійно еволюціонує під впливом колективних рішень аудиторії.

4.2 Захист розробленого програмного забезпечення

Забезпечення захисту алгоритму сервісу підтримку колективно-конкурентного планування створення відео-контенту є дуже важливим аспектом через те, що система оперує персональними даними користувачів. Основним механізмом захисту інформації, яка передається між сервером та клієнтом є протокол HTTPS, який передбачає обов’язкове використання протоколу TLS. TLS шифрує дані, що унеможлиблює їхнього перехоплення третьою стороною.

Для автентифікації використовується стандарт JSON Web Token (JWT). Цей стандарт дозволяє реалізувати сервіс таким чином, що серверу не потрібно зберігати інформацію про сесії у пам’яті, що підвищує стійкість системи. А також захищає додаток від доступу несанкціонованих користувачів до можливостей сервісу і до даних, якими оперує цей сервіс.

Для захисту прав інтелектуальної власності було обрано модель відкритого розповсюдження з контрольованим обмеженням – ліцензія Creative Commons. Основні умови обраної ліцензії:

- BY: обов’язкове зазначення авторства при будь-якому використанні коду.
- NC: Не можна використовувати програму для комерційних цілей.
- SA: Похідна робота повинна розповсюджуватись на умовах тієї ж самої ліцензії, що й оригінал.

Ліцензія була обрана з причин розповсюдженості в світі, в тому числі в Україні та гнучкого набору умов, які дозволяють популяризувати та поширювати продукт і водночас залишати ключові права за собою.

5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Для забезпечення стабільної та ефективної роботи розробленого веб-сервісу у реальних умовах промислової експлуатації було визначено перелік мінімальних технічних вимог до програмного та апаратного забезпечення:

- Серверна частина: середовище виконання Node.js версії 18.0 або вище, а також потребує пакетний менеджер npm для керування зовнішніми бібліотеками.

- База даних: робота з даними передбачає наявність встановленої та належним чином налаштованої СКБД PostgreSQL (або локальна desktop версія або з використанням хмарного рішення Supabase).

- Клієнтська частина: є крос-платформною і потребує лише сучасний веб-браузер, наприклад: Google Chrome, Mozilla Firefox, Opera із обов'язковою підтримкою стандартів HTML5 та протоколу WebSockets.

Процес впровадження системи складається з наступних кроків:

- Клонування проекту: відкрити систему контролю версій Git та отримати вихідний код сервісу з репозиторію в робочу директорію сервера.

- Налаштування середовища (.env): створення файлу конфігурації для конфіденційних даних таких як:

- а) рядок підключення до бази даних,
- б) секретний ключ підпису JWT-токенів.

- Встановлення залежностей: за допомогою команди npm install, яка автоматично завантажує та інсталує всі програмні залежності, необхідні для роботи frontend та backend частин.

- Ініціалізація бази даних: передбачає створення необхідних таблиць та зв'язків згідно з розробленою схемою, для коректного оперування замовленням та профілями користувачів.

- Запуск системи: для переведення сервісу в активний стан використовуються команди npm run build, яка компілює вихідний код у

Вим.	Дир.	№ докум.	Підпис	Дата

ВКРБ-122.26.0023.00.00.ПЗ

оптимізований для виконання пакет, та `npm run start` для активації серверного процесу.

Результатом успішного розгортання є виведення інтерфейсу вітальної сторінки, яка зображена на рисунку 5.1 і слугує точкою входу для відвідувачів. Подальша робота сервісу демонструє гнучкість управління чергами для стрімера та інтуїтивно зрозумілий процес замовлення відео для глядача, приклади яких наведені на рисунках 5.2 та 5.3 відповідно

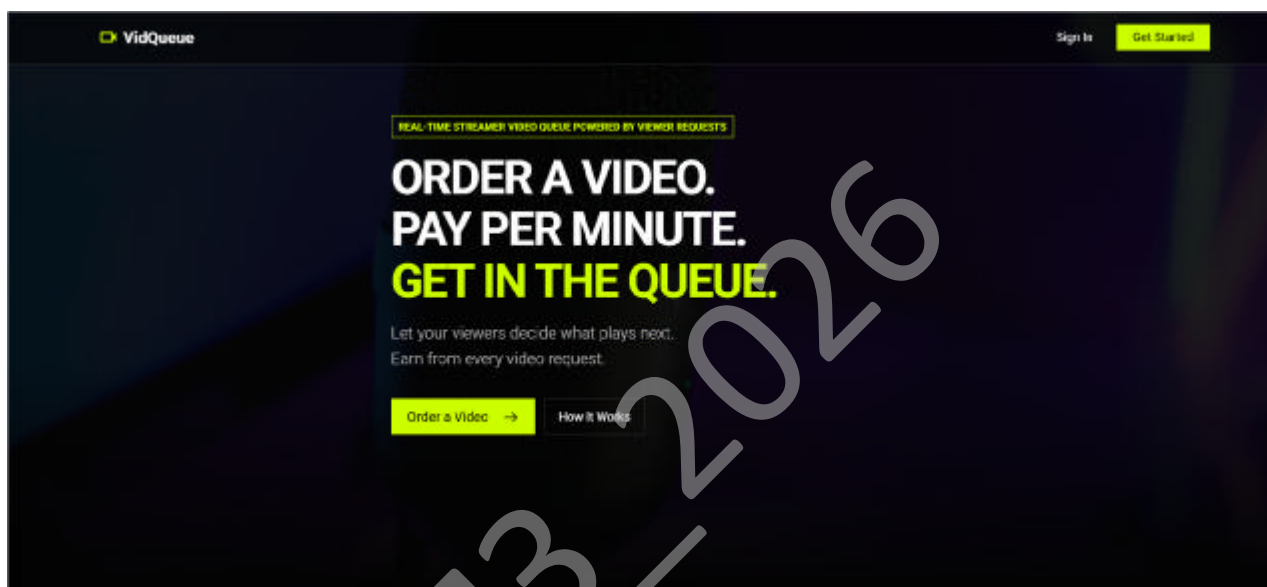


Рисунок 5.1 – Landing сторінка сервісу

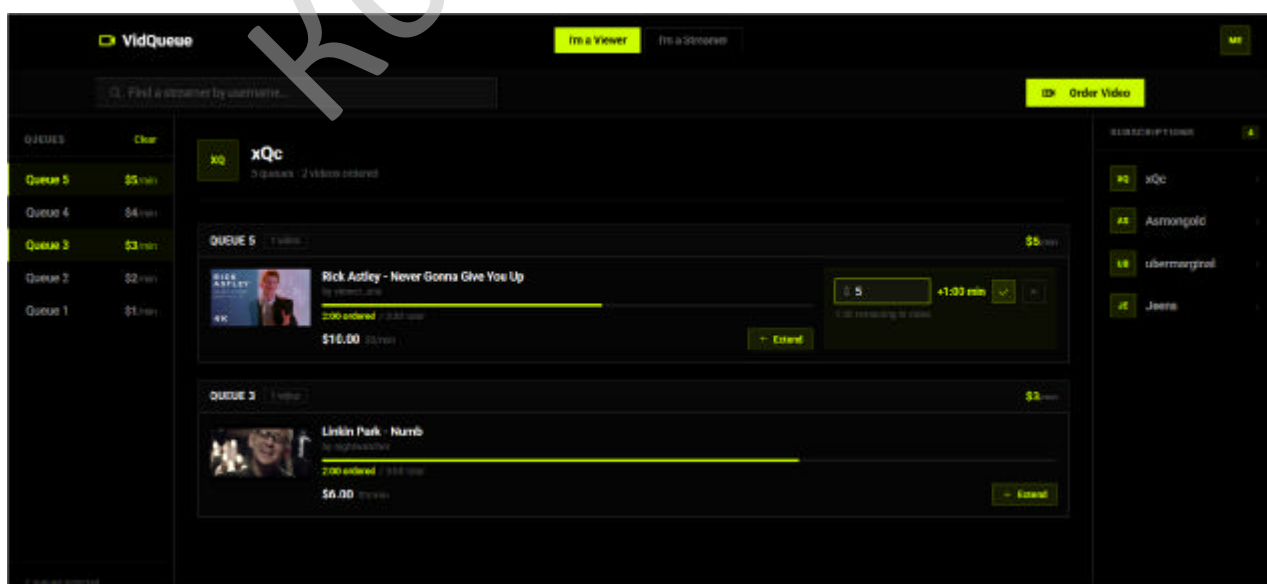


Рисунок 5.2 – Приклад роботи сервісу зі сторони глядача.

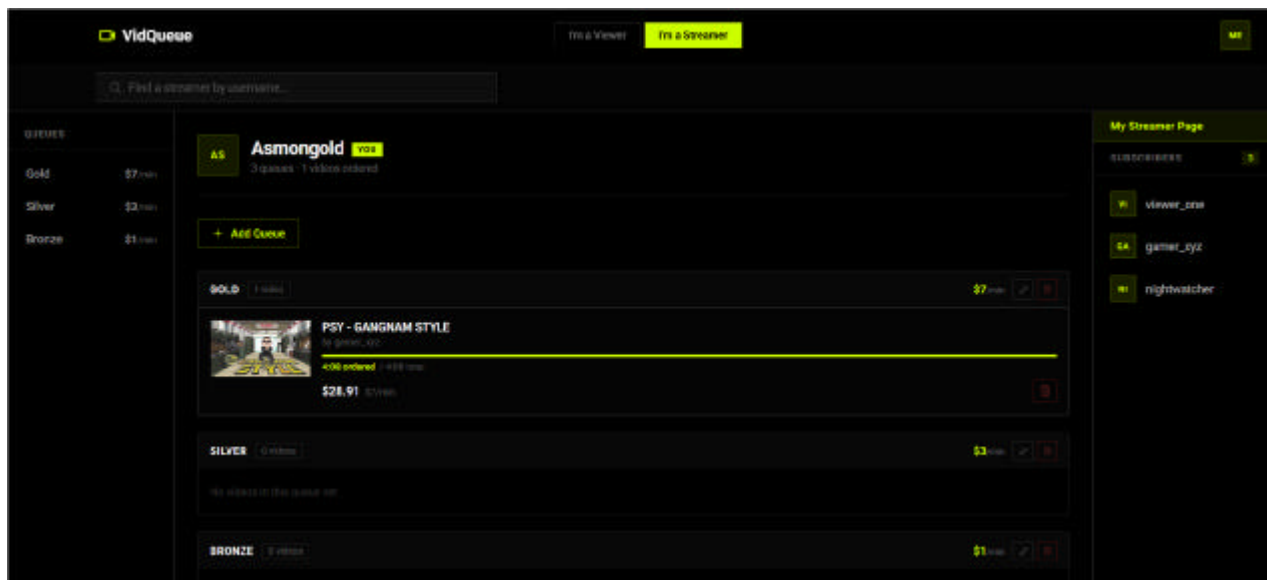


Рисунок 5.3 – Приклад роботи сервісу зі сторони стрімера.

К6ПЗ_2026

Вим	Дов	№ докум	Підпис	Дата

ВКРБ-122.26.0023.00.00.ПЗ

Арк.

48

6 ОСНОВНІ ВИСНОВКИ

У результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти було розроблено алгоритм сервісу підтримки колективно-конкурентного планування створення відео-контенту.

Для вирішення задачі було проведено:

- Аналіз існуючих систем підтримки колективно-конкурентного планування створення відео-контенту та виявлено їх обмеженість і недостатнє вирішення проблеми та підтверджено актуальність розробки через наявність “ручних” розробок систем планування у стрімерів, які потребували автоматизації та зручного інтерфейсу.

- Проектування алгоритму сервісу підтримки колективно-конкурентного планування створення відео-контенту.

- Програмна реалізація алгоритму сервісу підтримки колективно-конкурентного планування створення відео-контенту.

Розроблені під час виконання кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання підтримки колективно-конкурентного планування створення відео-контенту.

Розроблена система використовує об’єктно-орієнтований підхід, що робить її зручною для підтримки та масштабування, що відповідає сучасним галузям розробки програмного забезпечення.

Сервіс має простий та сучасний інтерфейс користувача, написаний за допомогою фреймворку React, що забезпечує високу швидкість відповіді та обробки даних.

Алгоритм сервісу підтримки колективно-конкурентного планування створення відео-контенту був розроблений за допомогою мови програмування TypeScript. Ця типізована мова програмування дозволила ефективно

Вим.	Дир.	№ докум.	Підпис	Дата

розробляти веб-додаток через свою простоту та уніфікованість як і для серверної частини додатку, так і для клієнтської частини програми.

Надана уся необхідна інформація та рекомендації для встановлення та експлуатації реалізованої системи.

Створений сервіс підтримки колективно-конкурентного планування створення відео-контенту, в цілому повністю відповідає вимогам технічного завдання, а також має потенціал для подальшого вдосконалення, враховуючи все більшу актуальність систем для планування створення “реакційного” контенту.

КБПЗ_2026

Вим	Дру	№ докум	Підпис	Дата

ВКРБ-122.26.0023.00.00.ПЗ

Арк.

50

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. U. S. Streaming Market 2026: Monetization, Payments, AI and Hybrid Models Reshape a Mature Digital Video Ecosystem. *yStats.com*. URL: <https://www.ystats.com/post/u-s-streaming-market-2026-monetization-payments-ai-and-hybrid-models-reshape-a-mature-digital-video-ecosystem> (дата звернення: 12.03.2026).

2. Як OTT-платформи змінюють медіаіндустрію й рекламний ринок в Україні. *Детектор медіа*. URL: <https://detector.media/rinok/article/237252/2025-01-20-yak-ott-platformy-zminyuyut-mediaindustriyu-y-reklamnyy-rynok-v-ukraini/> (дата звернення: 12.03.2026).

3. СТРИМІНГОВІ СЕРВІСИ В УКРАЇНІ – КВІТЕНЬ – ТРАВЕНЬ 2025. *gemius*. URL: <https://gemius.com/ua/блог/streamingovi-servisi-v-ukrayini-kviten-traven-2025/> (дата звернення: 12.03.2026).

4. Стратегії гейміфікації у 2025 році: стимулювання залучення, як ніколи раніше. *Smartico | Офіційний сайт*. URL: <https://www.smartico.ai/uk/blog-post/gamification-strategies-in-2025> (дата звернення: 12.03.2026).

5. Priority queue. *Wikipedia*. URL: https://en.wikipedia.org/wiki/Priority_queue (дата звернення: 12.03.2026).

6. Чому OTT-платформи привертають усе більше уваги: ключові тренди доставки відео через інтернет. *Mind*. URL: <https://mind.ua/openmind/20302882-chomu-ott-platformi-privertayut-use-bilshe-uvagi-klyuchovi-trendi-dostavki-video-cherez-internet> (дата звернення: 12.03.2026).

7. StreamElements about. *StreamElements*. URL: <https://streamelements.com/about> (дата звернення: 14.03.2026).

8. StreamLabs Desktop. *StreamLabs*. URL: <https://streamlabs.com/desktop-app> (дата звернення: 14.03.2026).

9. Дивимось. Список замовлених відео стрімера *Ubermarginal*. URL: https://docs.google.com/spreadsheets/d/19F2Mrl8OOsogART4Vu_iR1xpO9AAthwt7LOWTfchgNo/edit?gid=0#gid=0 (дата звернення: 14.03.2026).

10. Media Request Overview and Guide. *StreamElements*. URL: <https://support.streamelements.com/hc/en-us/articles/10474426856722-Media-Request-Overview-and-Guide> (дата звернення: 10.03.2026).

11. Any difference between Streamlabs and StreamElements. *Reddit*. URL: https://www.reddit.com/r/Twitch/comments/hhhotv/any_difference_between_streamlabs_and/ (дата звернення: 11.03.2026).

12. Streamlabs Desktop (Media Share). *Streamlabs*. URL: <https://streamlabs.com/stream-widgets/media-share?srsId=AfmBOoq19pTeWJv2CKewof3ivtjEgH9jfsxuB7jPFWh5FRDofGguw7ET> (дата звернення: 11.03.2026).

13. streamlabs/desktop. *GitHub*. URL: <https://github.com/streamlabs/desktop?tab=GPL-3.0-1-ov-file> (дата звернення: 12.03.2026).

14. Дивимось. Список замовлених відео стрімера *Ubermarginal*. URL: https://docs.google.com/spreadsheets/d/19F2Mrl8OOsogART4Vu_iR1xpO9AAthwt7LOWTfchgNo/edit?gid=0#gid=0 (дата звернення: 14.03.2026).

15. Замовлення. Список замовлених відео стрімера *Aaskar*. URL: <https://docs.google.com/spreadsheets/d/1YgIL2n38rzQrMdbusGWFxWfUaMYybazGKbmNthdiW5A/edit?gid=0#gid=0> (дата звернення: 14.03.2026).

16. Алгоритм заміщення комірок пам'яті FIFO. *Wikipedia*. URL: https://uk.wikipedia.org/wiki/Алгоритм_заміщення_комірок_пам'яті_FIFO (дата звернення: 15.03.2026).

17. React. *Wikipedia*. URL: <https://uk.wikipedia.org/wiki/React> (дата звернення: 15.03.2026).

18. React. *React.dev*. URL: <https://react.dev/> (дата звернення: 15.03.2026).

19. Vue.js. *Wikipedia*. URL: <https://uk.wikipedia.org/wiki/Vue.js> (дата звернення: 15.03.2026).

Вим.	Док.	№ докум.	Підпис	Дата

20. AngularJS. *Wikipedia*. URL: <https://en.wikipedia.org/wiki/AngularJS> (дата звернення: 15.03.2026).
21. TypeScript. *TypeScript | Official Website*. URL: <https://www.typescriptlang.org/> (дата звернення: 16.03.2026).
22. How to use React with TypeScript. *Zignuts*. URL: <https://www.zignuts.com/blog/how-to-use-react-with-typescript> (дата звернення: 16.03.2026).
23. React Router. *W3Schools*. URL: https://www.w3schools.com/react/react_router.asp (дата звернення: 17.03.2026).
24. babel-plugin-react-compiler. *npm*. URL: <https://www.npmjs.com/package/babel-plugin-react-compiler?activeTab=readme> (дата звернення: 17.03.2026).
25. Sass. *Wikipedia*. URL: <https://uk.wikipedia.org/wiki/Sass> (дата звернення: 18.03.2026).
26. Препроцесори CSS: SCSS, LESS і Stylus. *FoxmindEd*. URL: <https://foxminded.ua/preprotsesory-css-scss-less-i-stylus/> (дата звернення: 18.03.2026).
27. Node.js. *Nodejs.org*. URL: <https://nodejs.org/en> (дата звернення: 19.03.2026).
28. Що таке Node.js простими словами. *DAN.IT*. URL: <https://dan-it.com.ua/uk/blog/hto-jeto-takoe-node-js-prostymi-slovami/> (дата звернення: 19.03.2026).
29. Performance comparison: Rust vs Node.js vs Go. *Dev.to*. URL: <https://dev.to/hamzakhan/rust-vs-nodejs-vs-go-performance-comparison-for-backend-development-2g69> (дата звернення: 20.03.2026).
30. Node.js. *Wikipedia*. URL: <https://en.wikipedia.org/wiki/Node.js> (дата звернення: 20.03.2026).
31. PostgreSQL. *Wikipedia*. URL: <https://uk.wikipedia.org/wiki/PostgreSQL> (дата звернення: 21.03.2026).

32. PostgreSQL: що це. *FoxmindEd*. URL: <https://foxminded.ua/postgresql-shcho-tse/> (дата звернення: 21.03.2026).

33. ACID. *Wikipedia*. URL: <https://uk.wikipedia.org/wiki/ACID> (дата звернення: 22.03.2026).

34. Prisma ORM: brief overview. *Dev.to*. URL: <https://dev.to/sandrockjustin/the-prisma-orm-a-brief-overview-and-introduction-353m> (дата звернення: 22.03.2026).

35. Is Prisma ORM better than Parse ORM?. *Medium*. URL: <https://medium.com/@dileep18052001/is-prisma-orm-better-than-parse-orm-f93c4017bd35> (дата звернення: 23.03.2026).

36. Best IDEs code editor for developers in 2026. *Dev.to*. URL: https://dev.to/md8_habibullah/best-ides-code-editor-for-developers-in-2026-40i1 (дата звернення: 23.03.2026).

37. WebSockets: навіщо потрібні та як з ними працювати. *ProIT*. URL: <https://proit.ua/websockets-navishcho-potribni-ta-iak-z-nimi-pratsiuvati-2/> (дата звернення: 24.03.2026).

38. WebSocket handshaking explained. *Medium*. URL: <https://medium.com/@nile.bits/websocket-handshaking-explained-understanding-the-key-to-real-time-communication-52d631f84c0c> (дата звернення: 24.03.2026).

39. JSON Web Token. *Wikipedia*. URL: https://uk.wikipedia.org/wiki/JSON_Web_Token (дата звернення: 25.03.2026).

40. oEmbed. *oEmbed.com*. URL: <https://oembed.com/> (дата звернення: 25.03.2026).

41. Що таке REST API і як він працює. *DAN.IT*. URL: <https://dan-it.com.ua/uk/blog/shho-take-rest-api-i-yak-vin-praczuuye/> (дата звернення: 26.03.2026).

42. Що таке REST API. *FoxmindEd*. URL: <https://foxminded.ua/shcho-take-rest-api/> (дата звернення: 26.03.2026).

Вим	Дир	№ докум	Підпис	Дата

43. REST API: що це, як працює, переваги і приклади. *GoIT Global*. URL: <https://goit.global/ua/articles/rest-api-shcho-tse-iaak-pratsiuiie-perevahy-i-pryklady/> (дата звернення: 27.03.2026).

44. HTTPS: що це таке. *Portaone Education*. URL: <https://education.portaone.com/https-що-це-таке/> (дата звернення: 27.03.2026).

45. Transport Layer Security. *Wikipedia*. URL: https://uk.wikipedia.org/wiki/Transport_Layer_Security (дата звернення: 28.03.2026).

46. Що таке інтернет: хто його винайшов і як він працює. *IPnet*. URL: <https://ipnet.ua/blog/shcho-take-internet-khto-ioho-vynaishov-i-iaak-vin-pratsiuiie> (дата звернення: 28.03.2026).

47. Інтернет. *Wikipedia*. URL: <https://uk.wikipedia.org/wiki/Інтернет> (дата звернення: 29.03.2026).

48. Протокол TCP/IP або як працює інтернет. *QALight*. URL: <https://qalight.ua/baza-znaniy/protokol-tcp-ip-abo-yak-praczyu%D1%94-internet-dlya-pochatkivcziv/> (дата звернення: 29.03.2026).

49. Що таке Landing Page. *Freehost FAQ*. URL: <https://freehost.com.ua/ukr/faq/wiki/chto-takok-landing-page/> (дата звернення: 30.03.2026).

50. Модальне вікно. *Wikipedia*. URL: https://uk.wikipedia.org/wiki/Модальне_вікно (дата звернення: 30.03.2026).

51. CC BY-NC-SA 4.0. *Creative Commons*. <https://creativecommons.org/licenses/by-nc-sa/4.0/deed.uk> (дата звернення: 3.04.2026).