Галицький фаховий коледж імені В’ячеслава Чорновола
відділення комп’ютерних технологій
циклова комісія інформатики та комп’ютерних дисциплін

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач відділенням
комп’ютерних технологій
Наталія СТЕФУРАК /_____/
підпис
«___» _____ 2024 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи
освітньо-професійного ступеня «фаховий молодший бакалавр»
зі спеціальності 122 «Комп’ютерні науки»
на тему: «Вебсистема для спільного перегляду та обговорення відеоконтенту»

Студент групи КН-41 Віталій РОЖЕЛЮК _____
(підпис)

Керівник проєкту Сиротюк НАТАЛІЯ _____
(підпис)

Консультанти:
з техніко-економічного
обґрунтування Любов МЕЛЕНЧУК _____
(підпис)

нормоконтролер Наталія КУЛЬЧИНСЬКА _____
(підпис)

Тернопіль – 2024

Галицький фаховий коледж імені В’ячеслава Чорновола

відділення комп’ютерних технологій

циклова комісія інформатики та комп’ютерних дисциплін

ЗАТВЕРДЖУЮ

Завідувач відділенням

комп’ютерних технологій

Наталія СТЕФУРАК /_____/

підпис

«___» _____ 2023 р.

ЗАВДАННЯ

на кваліфікаційну роботу

на здобуття освітньо-професійного ступеня «фаховий молодший бакалавр»

студенту Рожелюку Віталію Васильовичу

(прізвище, ім’я та по-батькові студента)

1. Тема роботи «Вебсистема для спільного перегляду та обговорення відеоконтенту»

затверджено наказом по коледжу від “___” _____ 2023 р., №___

2. Термін здачі студентом завершеного проєкту “___” _____ 2024 р.

3. Вихідні дані до роботи: Результати досліження додатків для спільного перегляду відеоконтенту, ринку. Аналіз технологій реалізації

4. Перелік питань, які повинні бути розроблені в роботі:

а) основна частина: аналіз предметної області та постановка завдань, проєктування системи, реалізація та тестування системи

б) техніко-економічне обґрунтування: аналіз ринку, обґрунтування витрат на проєктування, обґрунтування необхідності

5. Перелік графічного матеріалу: макети сторінок вебсайту

6. Консультанти проєкту: Меленчук Л.І.

Розділ	Консультанти	Підпис, дата	
		Завдання видано	Завдання прийнято
3 техніко-економічного обґрунтування	Меленчук Л.І. (вчена ступінь, звання П.І.Б. консультанта)		

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

№ п/п	Найменування етапу	Терміни	
		початку	завершення
1.	Вибір теми, ознайомлення з вимогами до кваліфікаційної роботи	23.11.2023	02.12.2023
2.	Огляд типових рішень. Написання розділу	03.12.2023	07.12.2023
3.	Дослідження технологій реалізації	08.12.2023	10.12.2023
4.	Розробка функціональних вимог до вебсистеми та робота над її структурою	11.12.2023	18.12.2023
5.	Встановелння та налаштування середовища розробки	19.12.2023	05.01.2024
6.	Проєктування вебсистеми, її функціоналу та інтерфейсу	06.01.2024	28.01.2024
7.	Реалізація на налаштування вебсистеми	29.01.2024	19.04.2024
8.	Тестування вебсистеми	20.04.2024	29.04.2024
9.	Робота над економічним розділом	30.04.2024	10.05.2024
10.	Оформлення пояснювальної записки	11.05.2024	16.06.2024
11.	Попередній захист кваліфікаційної роботи, робота над зауваженнями	17.06.2024	17.06.2024
12.	Підготовка до захисту кваліфікаційної роботи	18.06.2024	26.06.2024
13.	Захист кваліфікаційної роботи	27.06.2024	27.06.2024

Дата видачі “___” _____ 202_ р. Керівник _____/ Сиротюк Н.С.

Завдання прийняв до виконання _____/ Рожелюк В.В.

Реферат

Вебсистема для спільного перегляду та обговорення відеоконтенту. Кваліфікаційна робота. Рожелюк Віталій Васильович. Галицький фаховий коледж імені В'ячеслава Чорновола, відділення комп'ютерних технологій. Спеціальність 122 «Комп'ютерні науки», 2024. Сторінок – 68, рисунків – 12, додатків – 2.

Об'єкт дослідження – вебсистеми для спільного перегляду відеоконтенту.

Метою роботи є пошук та аналіз існуючих рішень, визначення їй переваг та недоліків.

Завданням роботи є розробка системи для спільного перегляду та обговорення відеоконтенту.

Для реалізації даної вебсистеми було використано низку програмних засобів, інструментів та бібліотек, доступних у мовах програмування Python та JavaScript. Середовищем для реалізації роботи було обрано текстовий редактор NeoVim.

ВЕБСИСТЕМА ДЛЯ СПІЛЬНОГО ПЕРЕГЛЯДУ ТА ОБГОВОРЕННЯ ВІДЕОКОНТЕНТУ, PYTHON, DJANGO, CHANNELS, DAPHNE, JAVASCRIPT, NEOVIM.

Abstract

Web system for joint viewing and discussion of video content. Qualification work. Rozheliuk Vitalii Vasyliovych. Halych Vocational College named after Vyacheslav Chornovol, department of computer technologies. Specialty 122 "Computer Sciences", 2024. Pages – 68, figures – 12, appendices – 2.

The object of the research is web systems for joint viewing of video content.

The purpose of the work is to find and analyze existing solutions, to determine their advantages and disadvantages.

The task of the work is to develop a system for joint viewing and discussion of video content.

A number of software tools, tools and libraries available in the Python and JavaScript programming languages were used to implement this web system. The text editor NeoVim was chosen as the environment for the implementation of the work.

WEB SYSTEM FOR COMMON VIEWING AND DISCUSSION OF VIDEO CONTENT, PYTHON, DJANGO, CHANNELS, DAPHNE, JAVASCRIPT, NEOVIM

ЗМІСТ

Скорочення та умовні познаки 6
Вступ 7
1 Аналіз предметної області та постановка завдань 9
1.1 Опис предметної області 9
1.2 Аналіз наявних рішень 10
1.3 Аналіз вимог до програмного засобу та постановка завдання 14
2 Проєктування системи 16
2.1 Проєктування структури проєкту 16
2.2 Проєктування серверної частини 19
2.3 Проєктування клієнтської частини 21
2.4 Проєктування візуального інтерфейсу 21
3 Реалізація та тестування системи 25
3.1 Опис необхідного системного, прикладного та інструментального програмного забезпечення 25
3.2 Реалізація 37
3.3 Тестування системи 42
4 Техніко-економічне обгрунтування 48
4.1 Аналіз ринку 48
4.2 Розрахунок витрат на проєктування 49
4.3 Обґрунтування необхідності 52
Висновки 54
Перелік джерел посилання 56
Додатки 57

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

HTML – HyperText Markup Language

CSS – Cascading Style Sheets

ASGI – Asynchronous Server Gateway Interface

Redis – Remote Dictionary Server

Git – Global Information Tracker

HTTP – Hypertext Transfer Protocol

JS – JavaScript

MVP – Minimum Viable Product

CI/CD – Continuous Integration/Continuous Deployment

SSL – Secure Sockets Layer

DB – Database

API – Application Programming Interface

ВСТУП

Сучасні вебдодатки стають все більш важливим інструментом для різноманітних сфер діяльності, від бізнесу до освіти та побутових потреб користувачів. Швидкий розвиток інтернет-технологій та зростання вимог до зручності і ефективності використання зумовили необхідність у створенні вебсистем з високим рівнем функціональності та користувацької зручності.

У сучасному світі цифрових технологій відеоконтент займає одну з центральних позицій у сфері розваг, освіти та комунікації. З розвитком високошвидкісного інтернету та поширенням мобільних пристроїв, відео стало доступним будь-де і будь-коли. Водночас зростає потреба у створенні платформ, які дозволяють користувачам не тільки споживати відеоконтент, але й взаємодіяти з ним, ділитися враженнями та обговорювати переглянуте. Тема цієї кваліфікаційної роботи присвячена розробці вебсистеми для спільного перегляду та обговорення відеоконтенту.

Актуальність розробки такої системи обумовлена кількома факторами. По-перше, сучасні платформи для перегляду відео, такі як YouTube, Netflix та інші, хоча і надають можливість коментування відео, але часто не забезпечують синхронний перегляд та обговорення у режимі реального часу. Це особливо важливо для освітніх проєктів, де спільний перегляд відеоматеріалів може сприяти кращому засвоєнню матеріалу через колективне обговорення. По-друге, зростає попит на інтерактивні інструменти для дистанційного спілкування, що стало особливо актуальним під час пандемії COVID-19, коли багато людей змушені були працювати та навчатися дистанційно.

Основною метою цієї роботи є створення вебсистеми, яка дозволить користувачам спільно переглядати відео в режимі реального часу та обговорювати його через інтегрований чат. Система повинна підтримувати синхронізацію відео для всіх учасників сеансу, надавати можливість ставити

відео на паузу, перемотувати його, а також обмінюватися повідомленнями у чаті. Така платформа стане корисним інструментом для різноманітних аудиторій: від друзів, які хочуть разом подивитися фільм, до викладачів та студентів, які використовують відео для навчання.

У даній роботі буде розглянуто існуючі рішення у цій сфері, визначено їх переваги та недоліки, а також представлено власну концепцію та архітектуру вебсистеми. Особлива увага приділяється питанням синхронізації відео, забезпеченню стабільної роботи чату та зручності користувацького інтерфейсу. У завершенні роботи буде наведено результати тестування системи та зроблено висновки щодо її ефективності та подальших перспектив розвитку.

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАНЬ

1.1 Опис предметної області

Віддалена комунікація набуває все більшого значення, і вебсистема для спільного перегляду та обговорення відеоконтенту відіграє ключову роль у сприянні ефективній взаємодії та спільній роботі в онлайн-середовищі. Цей інструмент створює унікальну можливість для користувачів спільно переглядати відеоматеріали та обговорювати їх в реальному часі, зменшуючи відстань між учасниками та підвищуючи ефективність співпраці. Актуальність, проблематика та можливі сфери використання цієї захоплюючої технології заслуговують на детальний розгляд.

Вебсистема для спільного перегляду та обговорення відеоконтенту є високотехнологічним рішенням, яке впроваджується в умовах постійно зростаючої потреби в зручних та ефективних засобах віддаленої комунікації. Завдяки інноваційним можливостям, вона стає каталізатором для розвитку колективної взаємодії, особливо в умовах віддалених робочих середовищ.

Система вирішує основні проблеми сучасного спілкування, пропонуючи не тільки синхронний перегляд відеоматеріалів, але й створюючи інтерактивне середовище для обговорення. Це рішення дозволяє подолати дистанцію між учасниками, забезпечуючи відчуття спільності та взаємодії навіть у віртуальному просторі.

Актуальність визначається зростаючою необхідністю в ефективній дистанційній взаємодії в таких сферах, як робочі команди, освітні установи та бізнес-проєкти. Система стає важливим інструментом для підтримки цих процесів, забезпечуючи якісну і відзначену часом взаємодію, незалежно від місця перебування учасників.

Однією з ключових проблем, які вирішує платформа, є надання учасникам зручного механізму для синхронного коментування відео. Забезпечуючи доступ до інструментів реального часу для обговорення та

анотування відеоматеріалів, вона усуває обмеження традиційних форм віддаленої комунікації.

Можливості використання системи виявляються надзвичайно різноманітними: від дистанційного навчання та бізнес-конференцій до колективного творчого процесу, де кілька мозків об'єднуються для створення відеопроєктів. Вона стає важливим інструментом для забезпечення ефективної комунікації та спільної роботи у сферах медіа, розваг та багатьох інших галузях, де обговорення та аналіз відеоконтенту є критичними етапами.

1.2 Аналіз наявних рішень

У цьому розділі буде проведений аналіз існуючих вебсистем для спільного перегляду та обговорення відеоконтенту. Кожен аналог буде ретельно розглянутий, висвітлені його переваги та недоліки. Це дозволить отримати повний огляд доступних можливостей для користувачів, які цінують колективний перегляд відео та обговорення в режимі реального часу. Аналіз допоможе визначити, яка з платформ відповідає найкраще конкретним потребам та вимогам користувачів.

Watch2Gether – це вебсервіс, який дозволяє користувачам спільно переглядати відео та одночасно обговорювати його в режимі реального часу. Система надає можливість створювати віртуальні "кімнати", до яких можуть приєднатися інші користувачі. У цих кімнатах вони можуть додавати відео з різних джерел, таких як YouTube, Vimeo та інші, та спільно переглядати їх в синхронізованому режимі.

Основні функції Watch2Gether включають можливість обговорення відео за допомогою текстового чату, спільного вибору відео для перегляду та простий інтерфейс користувача. Варіанти користувачів можуть бути обмежені безкоштовною версією, але є також преміум-функції, які розширюють функціональність платформи.

Сторінка кімнати платформи зображена на рисунку 1.1

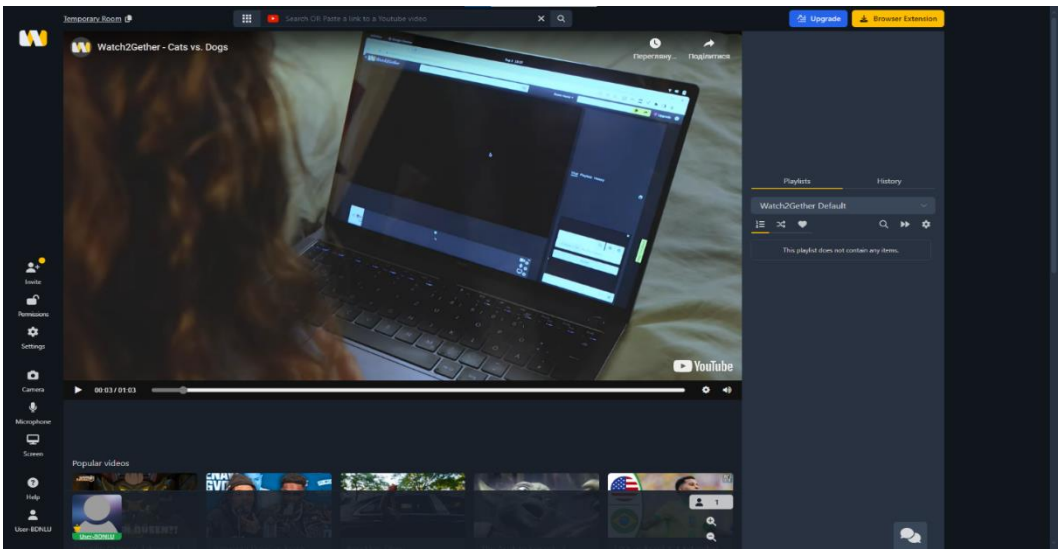

Рисунок 1.1 – Вигляд кімнати у Watch2Gether

Переваги:

– Чат.

– Плейлисти.

– Спільний контроль відтворення.

– Підтримка Youtube, Vimeo, Dailymotion, Soundcloud тощо.

– Можливість використання без реєстрації.

Недоліки:

– Обмежена функціональність безкоштовної версії, деякі додаткові можливості доступні лише в преміум-планах.

– Навантажений зайвою інформацією.

– Складний для першого сприйняття інтерфейс.

WatchParty.me – це вебплатформа, яка дозволяє користувачам спільно переглядати відеоконтент в реальному часі, обговорювати його та спілкуватися з іншими учасниками. Основна ідея полягає в тому, щоб люди могли створювати "вечірки" (watch parties) та запрошувати друзів або інших користувачів до спільного перегляду відео. Платформа працює через веббраузер, що робить її доступною для широкого кола користувачів без необхідності встановлення додаткового програмного забезпечення.

Сторінка кімнати платформи зображена на рисунку 1.2

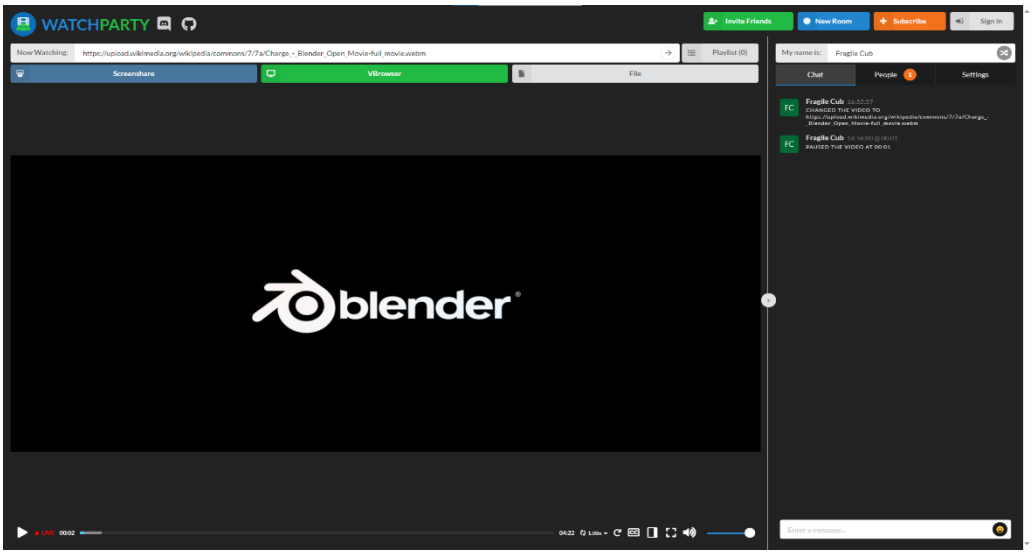

Рисунок 1.2 – Вигляд кімнати у WatchParty

WatchParty.me надає можливість користуватися платформою без створення облікового запису, що полегшує процес створення та приєднання до вечірок. Однак, деякі функції можуть бути обмежені для незареєстрованих користувачів.

Переваги:

– Чат.
– Відео Чат.
– Плейлисти.
– Спільний контроль відтворення.
– Vbrowser.
– Можливість ділитися екраном.
– Можливість використання без реєстрації.

Недоліки:

– Підтримка лише YouTube.
– Навантажений зайвою інформацією.
– Складний для першого сприйняття інтерфейс.

Syncplay– це програмний інструмент, який дозволяє користувачам синхронізувати відтворення медіафайлів в реальному часі, навіть якщо вони розташовані на різних пристроях та в різних мережах. Основна ідея Syncplay полягає в тому, щоб дозволити групі користувачів переглядати відео або слухати аудіо файл одночасно, забезпечуючи синхронізацію відтворення усіх учасників.

Основні особливості Syncplay включають можливість відтворення різних типів медіафайлів (відео, аудіо) з різних джерел, таких як локальні файли або онлайн-платформи. Програма також надає можливість обговорювати вміст за допомогою текстового чату.

Syncplay дозволяє користувачам створювати власні кімнати для перегляду та запрошувати інших користувачів до участі. Це робить його зручним інструментом для дружб, які хочуть спільно переглядати фільми або слухати музику, знаходячись в різних місцях.

Рисунок 1.3 – Syncplay

Переваги:

– Відкритий код.
– Спільний контроль відтворення.

Недоліки:

– Складність налаштування.
– Немає підтримки інших сервісів.

Після дослідження наявних систем було проведено аналіз за певними критеріями, який зображений в таблиці 1.1.

Таблиця 1.1 – Аналіз систем

Назва / Критерій	WatchParty	Watch2Gether	Syncplay
Вебзастосунок	+	+	-
Заміна відеофалу	+	+	+
Зупинка відео	+	+	+
Зміна часу відео	+	+	+
Зручний інтерфейс	+/-	+/-	-
Чат	+	+	-
Мінімалістичний інтерфейс	-	-	-

Розглянувши таблицю можа ствердити, що наявні системи не можуть забезпечити користувачів усіма потрібними функціями.

У свіх розглянутих засобів є такі негативна сторона: навантажений зайвою інформацією користувацький інтерфейс.

1.3 Аналіз вимог до програмного засобу та постановка завдання

Під час аналізу існуючих рішень, таких як Watch2Gether, WatchParty та Syncplay, визначено ряд переваг та недоліків. З метою розробки інноваційної вебсистеми, що враховує виявлені проблеми, ставляться конкретні завдання з розробки, включаючи забезпечення синхронного перегляду, функціоналу чату та підтримки різних джерел відеоконтенту. Цей процес спрямований на створення інтуїтивно зрозумілої та ефективної платформи для віддаленої спільної взаємодії.

Функціональні вимоги:

– Синхронізація відтворення відео для всіх учасників.

– Підтримка різних джерел відео таких як mp4, WebM та hls.

– Спільний контроль відтворення (пауза, відтворення, перемотування).

– Текстовий чат в реальному часі.

– Можливість ділитися емодзі.

– Простий та інтуїтивно зрозумілий інтерфейс.

Нефункціональні вимоги:

– Захист особистих даних користувачів.

– Конфіденційність спілкування.

– Швидка та плавна синхронізація відтворення.

– Можливість роботи на різних пристроях з різною пропускною здатністю.

– Можливість підтримувати велику кількість учасників.

– Можливість розширення функціональності.

Завдання включає в себе створення простого, мінімалістичного, не навантаженого зайвою інформацією, інтуїтивного інтерфейсу, який дозволить легко створювати віртуальні кімнати для перегляду та запрошувати інших учасників. Програмний продукт повинен підтримувати різні джерела відеоконтенту та надавати можливість обговорення через текстовий чат. Важливо забезпечити синхронізацію відтворення для усіх учасників, незалежно від їхніх технічних параметрів та мережевого з'єднання.

РОЗДІЛ 2 ПРОЄКТУВАННЯ СИСТЕМИ

Розвиток вебтехнологій та інтерактивних платформ для спільного використання медіа-контенту набуває великого значення. Відео є одним з найпопулярніших форматів контенту в Інтернеті, що створює попит на інструменти, які дозволяють не лише споживати вміст, але й обговорювати його разом з іншими користувачами в реальному часі.

У даному розділі будуть розглянуті основні аспекти проєктування системи, зокрема архітектурні рішення, вибір технологій, функціональні вимоги та особливості інтерфейсу користувача. Також будуть розглянуті методи та підходи, що використовуються для забезпечення безпеки, масштабованості та ефективності системи.

Проєктування вебсистеми для спільного перегляду та обговорення відеоконтенту має значний потенціал у сферах освіти, розваг та співпраці. Цей розділ відображає ключові аспекти роботи над проєктом та визначає основні напрямки подальших досліджень та розробки.

2.1 Проєктування структури проєкту

Вебсистема для спільного перегляду та обговорення відеоконтенту– це онлайн-платформа, яка дозволяє користувачам спільно переглядати відео та одночасно обговорювати його у реальному часі. Ця система матиме різноманітні функції, включаючи можливість синхронізації відтворення відео для усіх користувачів, чат для обговорення контенту, можливість реакцій (таких як емоційні реакції або коментарі) під час перегляду, можливість додавання анотацій або маркерів на відео тощо.

Проєктування і реалізація такої системи має велике значення, оскільки вона може використовуватися у різних сферах, таких як освіта, розваги, командна співпраця тощо. Правильно спроєктована система забезпечить користувачам зручний та ефективний інтерфейс для спільного перегляду та

обговорення контенту, а також забезпечить безпеку та масштабованість сервісу.

Успішна реалізація вимагає аналізу вимог, вибору відповідних технологій, розробки зручного інтерфейсу користувача, розробки ефективних алгоритмів синхронізації відео та обговорення, а також забезпечення безпеки даних та захисту приватності користувачів. Крім того, важливо правильно продумати архітектуру системи та її масштабованість для забезпечення стабільної роботи при великій кількості користувачів.

Перед початком проєктування потрібно обрати тип архітектури бекенду. Архітектура бекенду– це структура та організація серверної частини програмного забезпечення, яка відповідає за обробку запитів від клієнтської сторони, взаємодію з базою даних, обчислення бізнес-логіки та надання відповідей клієнтам.

В залежності від потреб проєкту та його специфікацій, можуть використовуватись різні типи архітектури бекенду, серед яких основні включають:

– Монолітна архітектура. У цьому типі архітектури всі компоненти бекенду розміщуються в єдиному програмному забезпеченні, яке розвивається, впроваджується та масштабується як єдина сутність. Це простий підхід до розробки, але може мати обмежену масштабованість та гнучкість.

– Мікросервісна архітектура. У цьому підході бекенд розділяється на невеликі автономні сервіси, які функціонують незалежно один від одного та взаємодіють через API. Кожен сервіс відповідає за обробку конкретної функціональності, що дозволяє досягти гнучкості та масштабованості, але при цьому збільшує складність управління.

Для реалізації проєкту було обрано монолітну архітектуру. Монолітна архітектура є традиційним підходом до побудови бекенду вебдодатків, де весь функціонал додатка об'єднується в одному цілому. У такій архітектурі весь код, який відповідає за обробку запитів, доступ до бази даних, бізнес-логіку, а

також взаємодію з іншими складовими додатка, розташований в єдиному монолітному додатку.

Основні переваги монолітної архітектури включають:

– Простота розробки та внесення змін: У монолітній архітектурі усі компоненти додатку взаємодіють між собою безпосередньо, що спрощує розробку та внесення змін.

– Легкість розгортання: Такий додаток можна розгорнути на одному сервері, що зменшує складність управління інфраструктурою.

– Простота моніторингу та налагодження: За рахунок того, що весь функціонал знаходиться в одному додатку, моніторинг та налагодження стає більш простим.

Проте, монолітна архітектура також має свої недоліки:

– Складність масштабування: Оскільки весь функціонал зосереджений в одному додатку, масштабування окремих компонентів може бути складним.

– Вертикальний розвиток: Зі зростанням функціональності додатка його розмір також зростає, що може призводити до складностей у розробці та підтримці.

– Одинична точка відмови: Якщо один компонент моноліту вийде з ладу, це може призвести до недоступності всього додатку.

Отже, у проєкті використовуватиметься монолітна архітектура, що дозволить спростити розробку та зменшити складність управління проєктом.

Для виконання роботи було обрано фреймворк Django. Django – високорівневий відкритий Python-фреймворк для розробки вебсистем [1].

У фреймворку Django, як і в багатьох інших вебфреймворках, реалізація додатку організована за концепцією шарової архітектури. У Django ця архітектура розділяє функціональність додатку на різні логічні шари, які виконують певні завдання. Основні шари в Django включають:

– Шар представлення (Presentation Layer): Цей шар відповідає за відображення інформації користувачам та обробку їх введення. В Django для цього використовуються шаблони (templates) для відображення HTML-сторінок, які відправляються користувачам, а також класи-контролери (views), які обробляють HTTP-запити та формують відповідь для користувача.

– Шар бізнес-логіки (Business Logic Layer): Цей шар включає всі компоненти, які відповідають за бізнес-логіку додатку, такі як моделі (models), які описують структуру даних і взаємозв'язки між ними, та класи-контролери (views), які виконують логічні операції над даними. Всі операції з базою даних, валідація даних та інші бізнес-правила зазвичай реалізовані в цьому шарі.

– Шар доступу до даних (Data Access Layer): Цей шар відповідає за взаємодію з базою даних. У Django цей шар представлений моделями (models), які описують структуру та взаємозв'язки даних, а також запитами до бази даних через ORM (Object-Relational Mapping). Кожен з цих шарів виконує певні функції та взаємодіє з іншими шарами, створюючи таким чином повноцінний вебдодаток.

Розділення функціональності на шари дозволяє підтримувати код додатку чистим, організованим та легко змінювати та розширювати.

2.2 Проєктування серверної частини

Проєктування WebSocket серверної частини включає створення асинхронного WebSocket сервера для обробки повідомлень у реальному часі. Проєктування WebSocket серверної частини включає декілька ключових компонентів: з'єднання, обробка повідомлень, взаємодія з Redis для зберігання стану відео та розповсюдження повідомлень у групі користувачів.

При розробці серверної частини з використанням WebSocket для управління відеовідтворенням в режимі реального часу, необхідно врахувати кілька ключових аспектів, а саме: вибір бібліотек для роботи з сокетами, вибір

засобу для збереження інформації станів відео та створення архітектури сервера.

Для реалізації даної задачі були обрані бібліотеки Channels та Daphne.

Django Channels – це бібліотека для Django, яка додає підтримку WebSockets та інших асинхронних протоколів, дозволяючи створювати в реальному часі додатки. Daphne – це HTTP, HTTP2 та WebSocket сервер, що обробляє запити від клієнтів і передає їх до Channels. Взаємодія цих модулів зображена на рисунку 2.1

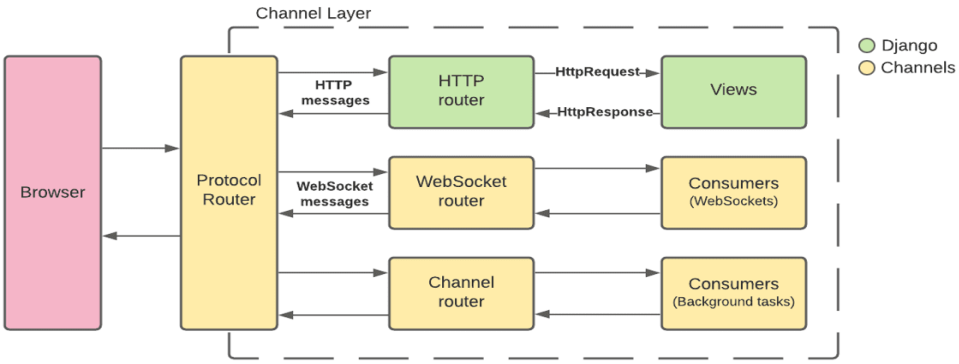

Рисунок 2.1– Взаємодія Django і Channels

Засобом, що буде швидко оновлювати інформацію про відео, було обрано Redis.

Redis використовується для зберігання та обміну інформацією між різними інстанціями WebSocket. Це забезпечує синхронізацію стану відео для всіх підключених клієнтів у реальному часі.

Основним компонентом архітектури серверної частини є WebSocket Consumer. Consumer Обробляє WebSocket запити від клієнтів, передає повідомлення між клієнтами через за допомогою Redis.

У програмному коді Consumer виступає в ролі класу, успадкованого від AsyncWebsocketConsumer модуля channels. Він містить метод “connect”, що викликається при новому підключенні та додає користувача до групи та відправляє початковий статус відео і метод disconnect, що видаляє користувача з групи при відключенні. Важливим є метод receive, який обробляє запити від

клієнтів та відповідно до запиту оновлює дані у Redis. Також клас містить методи для відправлень інформації до групи.

Така структура серверної частини дозволяє успішно синхронізувати відео серед користувачів і забезпечує функціональність чату у режимі реального часу.

2.3 Проєктування клієнтської частини

Проєктування клієнтської частини включає розробку функціональності для взаємодії з WebSocket сервером, обробки повідомлень та синхронізації відео серед користувачів у режимі реального часу.

При переході на сторінку кімнати відбувається підключення до сокета. Функція onmessage обробляє повідомлення, що надходять від WebSocket сервера, розрізняючи їх за типом і викликаючи відповідні функції для обробки кожного типу повідомлень.

Взаємодія з чатом відбувається за допомогою кнопки: при натисканні клавіші Enter або кнопки "send-message" відправляється повідомлення через WebSocket.

Також клієнтська частина обробляє події відео, такі як play та pause– відправляють статус відео при зміні стану, timeupdate– відправляє поточний час відтворення відео і seeked – відправляє новий час відтворення після перемотування.

Така структура клієнтської частини дозволяє синхронізувати відео серед користувачів і забезпечує функціональність чату у режимі реального часу.

2.4 Проєктування візуального інтерфейсу

Проєктування візуального інтерфейсу сайту є невід’ємною складовою розробки будь-якого сайту. Згідно з предметною областю сайт має містити мінімум дві сторінки, а саме сторінку для вибору кімнати, та сторінку кімнати.

Основною метою створення вебсистеми є надання клієнтам простого, але функціонально інтерфейсу для роботи з системою. Основні критерії оцінки зручності використання додатків включають:

Зовнішній вигляд додатку повинен бути розроблений так, щоб відповідати задачам, які клієнт має вирішувати за його допомогою.

Додаток повинен бути простим і зрозумілим, щоб нові користувачі могли швидко освоїти його та правильно використовувати без попереднього досвіду.

Додаток не повинен заважати роботі досвідчених користувачів, які використовують його протягом тривалого часу.

На сторінці вибору кімнати має бути поле для вводу username та назви кімнати. Макет сторінки зображений на рисунку 2.2

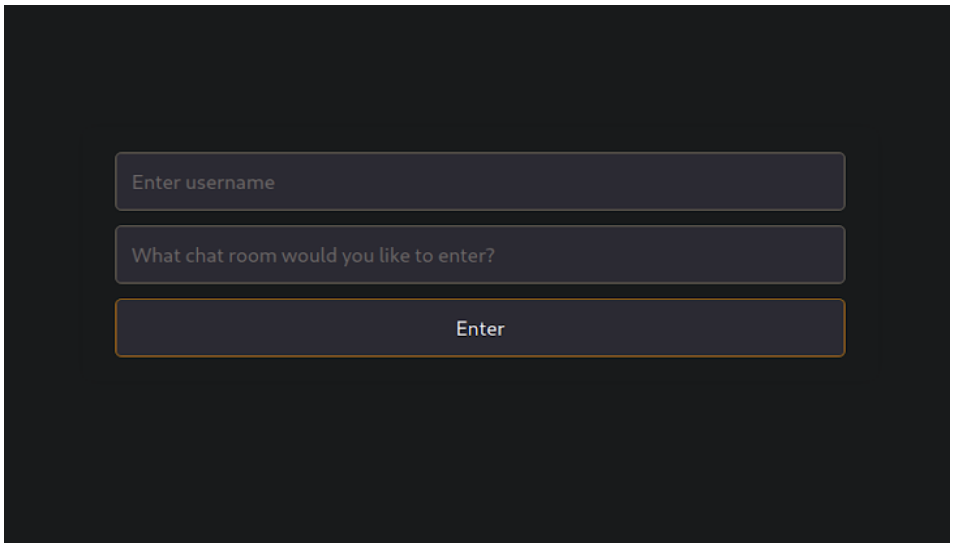

Рисунок 2.2 – Вибір кімнати

Сторінка кімнати має містити місце для відео, чату, та хедер в якому буде посилання на сторінку для вибору кімнати, елементи керування відео та кнопку надсилання повідомлення в чат. Макет сторінки зображений на рисунку 2.3.

Рисунок 2.3 – Кімната

Також важливо забезпечити коректне відображення сайту на різних пристроях та екранах. Для цього використовуються адаптивні (adaptive) та гнучкі (fluid) макети, медіа-запити та інші техніки для підтримки мобільної, планшетної та десктопної версій.

При проєктуванні візуального інтерфейсу сайту було вирішено використовувати колірну схему Gruvbox. Цей вибір був зроблений з кількох причин:

Gruvbox має ретро-естетику, яка створює теплу та приємну атмосферу завдяки своїм земляним тонам та м'яким контрастам.

М'які кольори Gruvbox менш стомлюють очі порівняно з яскравими та насиченими колірними схемами. Це особливо важливо для інтерфейсів, які користувачі будуть використовувати протягом тривалого часу.

Завдяки збалансованому контрасту між фоном та текстом, Gruvbox забезпечує високу читабельність, що важливо для користувацького досвіду.

Gruvbox підходить як для темних, так і для світлих тем, забезпечуючи гнучкість в дизайні. Це дозволяє створювати інтерфейси, які можуть легко адаптуватися до різних умов освітлення.

Отже, у цьому розділі було обрано тип архітектури вебсистеми, проєктування серверної та клієнтської частин та інтерфейсу системи. Розкрито основну інформацію, котру система має зберігати та обробляти, з'ясований основний функціонал, який повинен застосовувати вебзастосунок, для підтримання синхронного відтворення відео та чату у реальному часі.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Опис необхідного системного, прикладного та інструментального програмного забезпечення

Програмне забезпечення що потрібне для створення даного проєкту поділяється на три види: системне, прикладне і інструментальне (рисунок 3.1).

Рисунок 3.1 – Програмне Забезпечення

Системне програмне забезпечення налічує певну кількість програмних засобів, що допомагають комп'ютеру функціонувати: керують пам'яттю, даними, забезпечують працездатність інших програмних засобів, додатків тощо.

Операційна система включає в себе велику кількість програмних засобів, які використовуються в керуванні та взаємодії частин комп'ютера,управляє апаратними ресурсами комп'ютера та забезпечує середовище для виконання прикладних програм. Для реалізації даної кваліфікаційної роботи було використано операційну систему під назвою Arch Linux. Arch Linux – мінімалістичний, гнучкий дистрибутив Linux, оптимізований для архітектури x86-64 [2].

Прикладне програмне забезпечення призначене для виконання конкретних завдань, які корисні користувачам. Такі застосунки розробляються для виконання специфічних завдань чи функцій, таких як текстовий процесор для написання документів чи система керування базами даних.

Текстовий редактор – це програмне забезпечення, призначене для створення і редагування текстових файлів. Він забезпечує базову функціональність для введення, редагування та збереження тексту без складного форматування, що характерне для текстових процесорів . Текстові редактори зазвичай використовуються для написання коду, сценаріїв, конфігураційних файлів, а також для створення простих документів, де форматування не є важливим.

Текстові редактори можуть бути дуже простими, пропонуючи лише основні можливості, такі як введення тексту, вирізання, копіювання та вставка, або більш складними, з додатковими функціями, як підсвічування синтаксису, автозавершення, нумерація рядків та підтримка плагінів.

Neovim є одним із найкращих редакторів коду завдяки своїй швидкості, простоті налаштування та конфігурації [3]. (рис. 3.2). Він зберігає всі функції та можливості Vim, але додає численні покращення, спрямовані на підвищення зручності користування, продуктивності та розширюваності. Neovim створений з метою зробити процес редагування тексту більш ефективним та приємним для користувачів, що працюють з кодом та текстовими файлами.

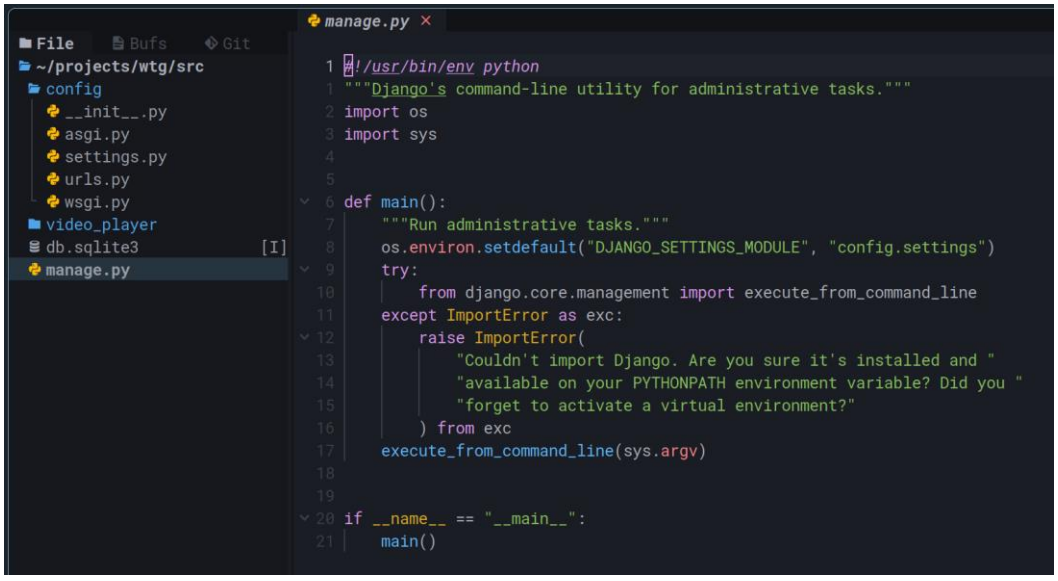

Рисунок 3.2 – Вигляд Neovim

Однією з основних переваг Neovim є його здатність до асинхронної роботи з плагінами, що дозволяє зменшити затримки та зробити роботу більш плавною. Він також має більш гнучку архітектуру, що спрощує створення та інтеграцію плагінів, дозволяючи користувачам налаштовувати редактор під свої потреби.

Neovim має такі вбудовані функції:

– Автодоповнення.
– Підсвічування синтаксису.
– Інтеграція з системами керування версіями.

Це робить його потужним інструментом для розробників та користувачів, які потребують швидкого та надійного текстового редактора.

Інструментальне програмне забезпечення – це тип програмного забезпечення, призначеного для розробки, тестування, діагностики та підтримки інших програм і систем. Воно забезпечує розробників необхідними інструментами для створення ефективного, надійного і безпечного коду. До інструментального програмного забезпечення належать мови програмування, компілятори, відлагоджувачі, а також різноманітні утиліти для тестування та аналізу коду. Інструментальне програмне забезпечення відіграє ключову роль у сучасному процесі розробки програмного забезпечення, надаючи розробникам всі необхідні засоби для створення, тестування та підтримки складних програмних систем. Розробку системи можна поділити на два етапи: написання серверної та клієнтської частин.

Клієнтська частина побудована на основі HTML, CSS і JavaScript. HTML відповідає за структуру вебсторінок, CSS – за їх зовнішній вигляд, а JavaScript – за поведінку та взаємодію.

HTML5 є основною мовою розмітки для створення вебсторінок і вебдодатків. Вона визначає структуру та зміст вебсторінок за допомогою різних елементів та їх атрибутів, які описують, як контент повинен відображатись у браузері.

CSS3 є останньою версією мови CSS, яка використовується для стилізації вебдокументів, написаних мовою HTML або інших мов розмітки. CSS визначає вигляд і розташування елементів на сторінці, дозволяючи контролювати кольори, шрифти, відступи, розміри , анімації та багато іншого.

JavaScript додає інтерактивність до вебсторінок. Він дозволяє реагувати на дії користувачів, відправляти асинхронні запити до сервера, маніпулювати DOM (Document Object Model) і динамічно оновлювати контент, такий як чат та відео.

WebSocket – це протокол, що призначений для обміну інформацією між браузером та вебсервером в режимі реального часу [4]. Це дозволяє обмінюватися даними в режимі реального часу, що робить WebSocket ідеальним для додатків, які потребують постійного оновлення даних, таких як чати, стрімінгові платформи та інші інтерактивні вебдодатки. Протокол WebSocket є частиною стандарту HTML5 і підтримується більшістю сучасних веббраузерів та вебсерверів.

Основні особливості WebSocket включають:

– Двонаправленість: WebSocket забезпечує можливість ініціювати передачу даних як від клієнта до сервера, так і в зворотному напрямку без необхідності чекати на відповідь на кожний окремий запит.

– Постійне з'єднання: Після встановлення з'єднання WebSocket залишається відкритим, що дозволяє миттєво відправляти та отримувати дані, зменшуючи затримки, пов'язані з відкриттям нових з'єднань.

– Легкість використання: На рівні реалізації WebSocket надає простий API для створення, підключення та обробки подій з'єднання, що спрощує розробку вебдодатків з реальним часом.

– Широке застосування: WebSocket використовується в багатьох вебдодатках, які потребують миттєвої взаємодії, таких як чати, колаборативні редактори, онлайн-ігри, фінансові платформи і багато іншого.

– Безпека: WebSocket підтримує захист даних через TLS (Transport Layer Security), що забезпечує шифрування даних та автентифікацію між клієнтом і сервером.

Серверна частина вебдодатка побудована на основі Python 3.12. з використанням фреймворку Django 4.2.9. Django є потужним і популярним фреймворком для веброзробки, який надає багато зручних інструментів для створення вебдодатків. Для реалізації WebSocket, який дозволяє встановлювати постійне двонаправлене з'єднання між клієнтом і сервером, використовується модуль channels 4.1.0 та модуль daphne 4.1.2.

Python – інтерпретована об'єктно-орієнтована мова програмування високого рівня із суворою динамічною типізацією [5]. Вона підтримує різноманітні парадигми програмування, включаючи процедурне, об'єктно-орієнтоване, функціональне та аспекто-орієнтоване програмування. Python широко використовується у веброзробці, наукових обчисленнях, автоматизації, штучному інтелекті та багатьох інших областях завдяки своїй потужності та багатофункціональності.

Django – це високорівневий вебфреймворк для Python, розроблений з метою спростити створення складних вебдодатків. Він надає готові рішення для багатьох типових завдань веброзробки, таких як робота з базами даних, обробка URL-адрес, автентифікація користувачів і управління сеансами. Django пропонує вбудовану підтримку адміністративного інтерфейсу, що дозволяє легко створювати і адмініструвати адмін-панелі для веб -додатків.

Однією з ключових особливостей Django є його вбудована архітектура MVC (Model -View-Controller), яка дозволяє розділити логіку додатку на компоненти моделі (доступ до даних), представлення (шаблони для відображення) і контролери (логіка обробки запитів). Він також пропонує потужну систему шаблонів, що дозволяє легко відокремити представлення від логіки програми.

Крім цього, Django активно розвивається і підтримується великою спільнотою розробників, що сприяє постійному вдосконаленню фреймворку і підтримці новітніх технологій веброзробки. Django добре підходить для будь-яких вебпроєктів, від малих сайтів до складних корпоративних систем, завдяки своїй потужності, гнучкості і широкому набору функціональних можливостей.

Channels– це бібліотека, яка розширює Django для підтримки асинхронних запитів, включаючи WebSocket та інші протоколи, які потребують постійного з'єднання.

Основні компоненти Django Channels:

– Channels: Канали – це шляхи, через які повідомлення приходять від одного місця до іншого. Вони можуть бути розглянуті як черги повідомлень.

– Consumers: Споживачі (consumers) – це асинхронні функції або класи, які приймають повідомлення з каналів і обробляють їх. Споживачі можуть бути WebSocket, HTTP, або інші види споживачів.

– Routing: Маршрутизація (routing) визначає, як споживачі підключаються до каналів. Це схоже на URL конфігурацію в традиційному Django додатку, але для каналів.

– Layer: Канальний шар (channel layer) – це абстракція для обміну повідомленнями між різними споживачами та екземплярами додатку. Redis використовується як бекенд для цього шару.

Daphne– це вебсервер WebSocket для Django, який використовується для обробки асинхронних запитів. Основні особливості Daphne:

– Підтримка протоколу ASGI: Daphne реалізує ASGI (Asynchronous Server Gateway Interface), який є стандартом для асинхронних Python-серверів, аналогічним WSGI для синхронних серверів.

– Обробка WebSocket: Daphne призначений спеціально для обробки WebSocket-з'єднань, що дозволяє йому ефективно управляти багатьма одночасними з'єднаннями.

ASGI (Asynchronous Server Gateway Interface)– це стандарт, розроблений для обробки асинхронних HTTP-запитів та інших асинхронних задач у Python. Він є розширенням стандарту WSGI (Web Server Gateway Interface), який використовується для синхронних вебзастосунків.

Основна різниця між ASGI і WSGI полягає в тому, що ASGI підтримує асинхронні сервери і дозволяє обробляти не тільки HTTP-запити, але й інші типи асинхронних запитів, такі як WebSocket, AMQP, MQTT тощо. Це робить ASGI ідеальним для використання в асинхронних вебдодатках, де потрібна підтримка реального часу і обробка багатьох одночасних з'єднань.

ASGI включає в себе специфікації для додатків (Application) і серверів (Server), які забезпечують стандартизований спосіб взаємодії між вебдодатками і серверами , що підтримують ASGI. Цей стандарт активно використовується в сучасних вебфреймворках для Python, таких як Django Channels, що дозволяє створювати потужні і ефективні асинхронні вебдодатки.

Redis – розподілене сховище пар ключ-значення, які зберігаються в оперативній пам'яті, з можливістю забезпечувати довговічність зберігання на бажання користувача [6]. Вона дозволяє зберігати і керувати різноманітними структурами даних, такими як рядки, списки, множини, хеші та інші, що робить її дуже гнучкою для різних застосувань.

Основна перевага Redis полягає у високій швидкості роботи, оскільки всі операції виконуються в оперативній пам'яті, що значно зменшує час доступу до даних. Це робить Redis ідеальним вибором для задач, що вимагають миттєвої відповіді, наприклад, кешування, управління сесіями користувачів, обробки черг повідомлень і аналітики в реальному часі.

Redis підтримує стійкість даних через різні механізми, такі як знімки бази даних і журнали змін, що дозволяє зберігати дані на диску і відновлювати їх у разі збою системи. Крім того, Redis забезпечує високу доступність і масштабованість через підтримку реплікації та кластеризації, що дозволяє

розподіляти дані по кількох вузлах і забезпечувати безперебійну роботу системи.

Завдяки своїй продуктивності, гнучкості і надійності, Redis став одним із найпопулярніших інструментів для зберігання даних і обробки інформації в реальному часі, використовуючись у багатьох високонавантажених додатках і системах.

У даній роботі Redis використовується для забезпечення передачі даних про відео та обміну повідомлень у чаті. Запуск Redis відбувається в Докері за допомогою команди docker run --rm -p 6379:6379 redis:7.

Docker – інструментарій для управління ізольованими Linux-контейнерами [7]. Основними компонентами Docker є Docker Engine, що дозволяє створювати та управляти контейнерами, Docker Hub для зберігання та обміну контейнеризованими додатками, і Docker Compose для оркестрації багатьох контейнерів.

Контейнери Docker використовують технологію контейнеризації для запуску програмного забезпечення у стандартизованому середовищі. Кожен контейнер містить програму разом з усіма необхідними залежностями, такими як бібліотеки і налаштування, що дозволяє їм працювати на будь-якій сумісній системі.

Git – розподілена система керування версіями файлів та спільної роботи[8]. Основна ідея полягає в тому, щоб кожен розробник мав повну копію історії змін проєкту, що дозволяє працювати над ним незалежно від централізованого сервера. Це забезпечує більшу надійність і гнучкість в порівнянні з традиційними централізованими системами керування версіями.

При роботі з Git, зміни в коді зберігаються у вигляді "знімків" (snapshots), які можна переглядати і повертатися до них у будь-який момент. Це дозволяє легко відстежувати всі зміни, зроблені в проєкті, і швидко знаходити та виправляти помилки. Крім того, кожен знімок містить

унікальний ідентифікатор, що допомагає уникати конфліктів між змінами різних розробників.

Git надає потужні інструменти для об'єднання різних гілок розробки. Це дозволяє розробникам працювати над різними функціональностями або виправленнями помилок одночасно, не заважаючи один одному. Гілки можуть бути об'єднані в основну гілку (main або master), коли зміни готові для інтеграції в основний код.

Одна з ключових особливостей Git – це його розподілений характер. Це означає, що кожен розробник має повну копію всього репозиторію, включаючи всю історію змін. Такий підхід дозволяє працювати в автономному режимі і синхронізувати зміни з іншими розробниками тільки тоді, коли це необхідно. Це також забезпечує додаткову надійність, оскільки навіть у випадку збою центрального сервера, кожен розробник має повну копію проєкту.

Git підтримує різноманітні робочі процеси, що робить його гнучким і придатним для різних типів проєктів та команд. Система також інтегрується з багатьма іншими інструментами, такими як системи безперервної інтеграції та розгортання (CI/CD), що робить її важливою частиною сучасної розробки програмного забезпечення.

Pre-commit – це фреймворк для автоматизації і перевірки змін в коді перед їх комітом в репозиторій. Його основна мета – забезпечити, щоб код відповідав певним стандартам і правилам перед тим, як він стане частиною основної гілки проєкту. Це досягається за допомогою запуску набору скриптів або гук-хуків на локальному комп'ютері розробника до того, як зміни будуть зафіксовані.

Pre-commit дозволяє автоматизувати процес перевірки коду, наприклад, аналіз статичних помилок, форматування коду, тестування і багато іншого. Це значно знижує ймовірність інтеграції помилок і недоліків у основний код, оскільки кожна зміна проходить попередню перевірку. Крім того, він

допомагає забезпечити дотримання єдиного стилю і стандартів написання коду в команді, що полегшує його читання і підтримку.

Фреймворк pre-commit інтегрується з Git і легко налаштовується за допомогою конфігураційного файлу. Цей файл містить список хуків, які мають бути виконані перед комітом, разом з їх параметрами. Це робить систему дуже гнучкою і дозволяє адаптувати її під конкретні потреби проєкту. Наприклад, можна налаштувати запуск лінтерів для перевірки стилю коду, інструментів для пошуку вразливостей або навіть автоматичне виправлення простих помилок.

За допомогою pre-commit можна забезпечити більш високий рівень якості коду, знизити кількість багів і підвищити продуктивність розробників. Це досягається завдяки автоматизації рутинних завдань і зменшенню часу на перевірку змін вручну. Pre-commit стає важливим інструментом у робочому процесі сучасних команд розробників, допомагаючи підтримувати високу якість коду і спрощувати процес його інтеграції.

Конфігурація pre-commit для даного проєкту наведена у лістингу 3.1

Лістинг 3.1 – Файл .pre-commit-config.yaml

```
repos:
 - repo: https://github.com/pre-commit/mirrors-mypy
     rev: "v1.8.0"
     hooks:
     - id: mypy
     additional_dependencies:
          - django-stubs
          - daphne
          - types-redis
     args: [--no-namespace-packages]
 - repo: https://github.com/psf/black
     rev: "23.12.1"
     hooks:
     - id: black
 - repo: https://github.com/pycqa/isort
```

```
    rev: "5.13.2"
    hooks:
    - id: isort
- repo: https://github.com/charliermarsh/ruff-pre-commit
    rev: "v0.1.13"
    hooks:
    - id: ruff
    args: [--fix, --exit-non-zero-on-fix]
```

Муру – це інструмент для статичної типізації в Python, який дозволяє перевіряти типи у вашому коді без виконання його. Використовуючи анотації типів, муру аналізує код і виявляє потенційні помилки типів, що допомагає уникнути багатьох помилок ще до запуску програми. Це особливо корисно в великих проєктах, де відстеження типів вручну може бути складним і схильним до помилок. Муру сприяє підвищенню надійності коду і робить його більш зрозумілим для інших розробників.

Black – це форматувач коду для Python, який автоматично приводить ваш код до єдиного стилю. Він суворо дотримується визначених правил форматування, що дозволяє зосередитися на логіці програми, а не на стилістичних питаннях. Використання Black спрощує командну роботу, оскільки усуває суперечки щодо стилю коду, забезпечуючи єдину структуру і форматування в усьому проєкті. Це також допомагає поліпшити читабельність коду і зменшує час на його огляд.

Isort – це інструмент для автоматичного сортування і упорядкування імпортів у Python-коді. Він допомагає підтримувати чистоту і порядок в імпортах, що робить код більш організованим і легшим для читання. Isort автоматично визначає і сортує імпорти за категоріями, такими як стандартні бібліотеки, сторонні пакети та локальні модулі, що забезпечує структурованість і узгодженість в коді. Це також знижує ризик виникнення конфліктів імпортів і полегшує навігацію по коду.

Ruff – це швидкий лінтер для Python, розроблений з акцентом на високу продуктивність і ефективність. Ruff перевіряє код на відповідність різним

правилам і стандартам , допомагаючи виявляти потенційні проблеми і недоліки. Завдяки своїй швидкості , Ruff дозволяє швидко аналізувати великі обсяги коду, що робить його корисним інструментом для постійного контролю якості. Використання Ruff допомагає підтримувати високі стандарти коду, вчасно виявляючи і виправляючи помилки.

Poetry – це інструмент для керування залежностями та пакетами в Python, який полегшує створення, управління та публікацію Python-проєктів. Основна мета Poetry – зробити процес роботи з залежностями та конфігураціями проєктів більш зручним і менш складним . Використовуючи Poetry, розробники можуть легко визначати залежності своїх проєктів , автоматично створювати і оновлювати файли конфігурації, а також встановлювати пакети.

Однією з ключових особливостей Poetry є його інтеграція з системою віртуальних середовищ Python, що дозволяє ізолювати залежності кожного проєкту, забезпечуючи , що вони не конфліктують між собою. Це особливо корисно, коли потрібно працювати з різними версіями одних і тих самих бібліотек у різних проєктах. Крім того, Poetry автоматизує процес створення та оновлення файлів конфігурації, таких як pyproject.toml, що значно спрощує керування проєктами.

Poetry також полегшує процес публікації пакетів у PyPI, надаючи прості команди для завантаження та розповсюдження пакетів. Це робить його корисним інструментом для розробників, які створюють та підтримують бібліотеки Python. Зручність і простота використання Poetry робить його популярним вибором серед розробників, які шукають ефективний спосіб управління залежностями і конфігураціями у своїх проєктах.

Poetry сприяє забезпеченню стабільності проєктів, завдяки точному контролю версій залежностей та автоматичному створенню файлів блокувань. Це допомагає уникнути проблем, пов 'язаних зі змінами в бібліотеках, що можуть вплинути на роботу проєкту. Poetry стає невід'ємною частиною

інструментарію сучасного Python-розробника, допомагаючи ефективно управляти залежностями, проєктами та процесом їх розповсюдження.

У лістингу 3.2 подано список усіх потрібних модулів. Завантажити їй можна за допомогою команди poetry install.

Лістинг 3.2 – Файл pyproject.toml

```
[tool.poetry.dependencies]
python = "^3.12"
Django = "4.2.9"
channels = "^4.1.0"
daphne = "^4.1.2"
channels-redis = "^4.2.0"
selenium = "^4.21.0"
redis = "^5.0.5"
asyncio = "^3.4.3"
types-redis = "^4.6.0.20240425"
[tool.poetry.group.dev.dependencies]
mypy = "^1.8.0"
django-stubs = "^4.2.7"
pre-commit = "^3.6.0"
[build-system]
requires = ["poetry-core"]
build-backend = "poetry.core.masonry.api"
[tool.mypy]
plugins = ["mypy_django_plugin.main"]
mypy_path = "./src"
ignore_missing_imports = true
[tool.django-stubs]
django_settings_module = "src.config.settings"
```

3.2 Реалізація

Усі користувачі додатку можуть створювати кімнати та приєднуватись до вже існуючих. Для цього на головній сторінці потрібно у формі ввести username та room_name. Форма для введення даних:

```
    <form id="chat-form">
      <input id="user-name-input" type="text" name="username"
size="100" required placeholder="Enter username" />
      <br/>
      <input id="room-name-input" type="text" name="roomname"
size="100" required placeholder="What chat room would you like to
enter?"/>
      <br/>
      <input id="room-name-submit" type="submit" value="Enter"
/>
    </form>
```

При натисканні на кнопку "room-name-submit" відбувається створення та перехід на сторінку кімнати. Перехід на сторінку кімнати відбувається за допомогою наступної реалізації:

```
    document.querySelector("#chat-form").onsubmit = function (e)
{
      e.preventDefault();
      var username = document.querySelector("#user-name-
input").value;
      var roomName = document.querySelector("#room-name-
input").value;
      window.location.href =
        "/room/" + roomName + "/" +
encodeURIComponent(username);
    };
```

Головною умовою функціонування додатку є з'єднання з WebSocket, яке відбувається одразу після переходу на сторінку кімнати. При з'єднанні з вебсокетом отримуються video media properties, такі як src, paused та currentTime. Дані отримуються шляхом надсилання запиту до Redis, поданого в лістингу 3.3, після чого надсилаються на клієнт.

Лістинг 3.3 – Отримання даних з Redis

```
    async def get_redis_video_status(self):
        """Get info about video from Redis"""
        redis_conn = redis.Redis(host="localhost", port=6379,
db=0)
        paused = await
redis_conn.get(f"{self.room_group_name}_video_paused")
        currentTime = await
redis_conn.get(f"{self.room_group_name}_video_currentTime")
        src = await
redis_conn.get(f"{self.room_group_name}_video_src")
        await redis_conn.close()
        return (paused, currentTime, src)
```

Клієнтська частина відслідковує основні події з відео, такі як play, pause, seeked , та при виконанні кожного з них робить відповідні запити на сервер за допомогою WebSocket, зображені у лістингу 3.4. Повний лістинг програмного коду клієнтської частини знаходиться в додатку А.

Лістинг 3.4 – Запити на сервер

```
video.addEventListener("play", function () {
  socket.send(
      JSON.stringify({
      type: "video.play",
      paused: 0,
      currentTime: video.currentTime,
      src: video.src,
      }),
  );
});
video.addEventListener("pause", function () {
  socket.send(
      JSON.stringify({
      type: "video.pause",
      paused: 1,
      currentTime: video.currentTime,
```

```
        src: video.src,
        }),
    );
});
video.addEventListener("seeked", function () {
    socket.send(
        JSON.stringify({
        type: "video.seeked",
        currentTime: video.currentTime,
        }),
    );
});
```

Сервер відрізняє тип запиту за допомогою ключа "type". Відтак якщо один із користувачів натисне на кнопку паузи, це відслідкується за допомогою video.addEventListener, відбудеться запит на сервер, що обробиться відповідною частиною розгалуження, поданою у лістингу 3.5, що надішле запити на інші клієнти. Отримавши запит, клієнт обробляє його відповідно до частини розгалуження, зображеної у лістингу 3.6, та функції, зображеної у лістингу 3.7. Повний лістинг програмного коду серверної частини знаходиться в додатку Б.

Лістинг 3.5– Обробка запиту на сервері

```
async def receive(self, text_data):
        try:
        text_data_json = json.loads(text_data)
        except json.JSONDecodeError:
        await self.send_error_message("Invalid JSON format")
        return
        if "type" not in text_data_json:
        await self.send_error_message("Invalid message type")
        return
        msg_type = text_data_json["type"]
        if msg_type == "video.pause":
```

```
            await
self.set_redis_video_paused(text_data_json["paused"])
            await self.channel_layer.group_send(
                  self.room_group_name,
                  {
                        "type": msg_type,
                        "paused": text_data_json["paused"],
                  },
            )
```

Лістинг 3.6– Обробка типу запиту на клієнті

```
socket.onmessage = function (event) {
  const data = JSON.parse(event.data);
  console.log(data);
  if (data.type === "video.pause" || data.type === "video.play") {
     handleVideoPaused(data);
```

Лістинг 3.7– Обробка запиту "pause"

```
function handleVideoPaused(data) {
  console.log(data.paused);
  if (data.paused === 1) {
     video.pause();
  } else if (data.paused === 0) {
     video.play();
  }
}
```

Функція що відповідає за збереження стану паузи для відео:

```
async def set_redis_video_paused(self, paused):
      """Set video.src to Redis"""
      redis_conn = redis.Redis(host="localhost", port=6379,
db=0)
```

```
            await
redis_conn.set(f"{self.room_group_name}_video_paused", paused)
            await redis_conn.close()
```

Згідно такого ж принципу здійснюється обробка усіх запитів користувачів кімнати.

3.3 Тестування системи

Тестування вебзастосунку є критично важливим з багатьох причин, що охоплюють різні аспекти розробки, безпеки, продуктивності та користувацького досвіду. Перш за все, тестування дозволяє виявити та виправити баги до того, як користувачі зіткнуться з ними. Це значно зменшує ризик виникнення серйозних проблем в подальшому, що може вплинути на репутацію компанії та довіру користувачів.

Продуктивність вебзастосунку також потребує ретельного тестування. Користувачі очікують, що застосунок буде працювати швидко та без збоїв. Тестування продуктивності допомагає визначити, як застосунок поводиться під великим навантаженням і чи здатний він витримати пікові періоди активності користувачів. Це включає стрес-тестування, навантажувальне тестування та тестування на масштабованість.

Користувацький досвід є ще одним важливим аспектом. Тестування забезпечує впевненість у тому, що вебзастосунок є інтуїтивно зрозумілим та зручним у використанні. Це включає перевірку коректності відображення на різних пристроях та браузерах, перевірку функціональності інтерфейсу, а також тестування на відповідність стандартам доступності.

Автоматизація тестування дозволяє зменшити витрати часу та ресурсів на повторювані перевірки, забезпечуючи швидке виявлення регресій після внесення змін до коду. Це сприяє підтримці високої якості коду та пришвидшує процес розробки.

Таким чином, тестування вебзастосунку є невід'ємною частиною процесу розробки, яка забезпечує надійність, безпеку, продуктивність та

високий рівень користувацького досвіду. Всі ці фактори в сукупності допомагають створити якісний продукт, що відповідає очікуванням користувачів та бізнес-вимогам.

Функціональне тестування та нефункціональне тестування є двома основними категоріями тестування програмного забезпечення, кожна з яких має свої цілі, методи та важливість.

Функціональне тестування спрямоване на перевірку того, чи працює програмне забезпечення відповідно до визначених вимог і специфікацій. Основна мета цього типу тестування – переконатися, що всі функціональні можливості, передбачені в додатку, працюють правильно. Функціональне тестування охоплює такі аспекти, як правильність обробки даних, взаємодія між компонентами, перевірка інтерфейсу користувача. Це тестування зазвичай включає створення тестових сценаріїв, які охоплюють всі можливі варіанти використання програми, включаючи як позитивні, так і негативні випадки. Приклади функціонального тестування включають модульне тестування, інтеграційне тестування, системне тестування та приймальне тестування

Звідси було проведено функціональне тестування вебзастосунку, використовуючи Selenium для автоматизації браузера та перевірки взаємодії користувачів з чат-застосунком. Тестування включає два тести: Тестування повідомлень у межах однієї кімнати (лістинг 3.7) та тестування повідомлень у різних кімнатах (лістинг 3.8). Виконання тестів подано на рисунку 3.3.

Під час першого тесту відкриваються два вікна браузера, кожне з яких заходить в одну й ту саму кімнату чату. Один користувач надсилає повідомлення, і тест перевіряє, що це повідомлення з'являється в обох вікнах.

Лістинг 3.7– Одна кімната

```
        def
test_when_chat_message_posted_then_seen_by_everyone_in_same_room
(self):
        try:
             self._enter_room("user_1", "room_1")
```

```
                self._open_new_window()
                self._enter_room("user_2", "room_1")
                self._switch_to_window(0)
                self._send_msg("hello")
                WebDriverWait(self.driver, 2).until(
                     lambda _: "hello" in self._chat_log_value,
                     "Message was not received by window 1 from
window 1",
                )
                self._switch_to_window(1)
                WebDriverWait(self.driver, 2).until(
                     lambda _: "hello" in self._chat_log_value,
                     "Message was not received by window 2 from
window 1",
                )
           finally:
                self._close_all_new_windows()
```

Під час другого тесту відкриваються два вікна браузера, кожне з яких заходить в різні кімнати чату. Один користувач надсилає повідомлення в свою кімнату, і тест перевіряє, що це повідомлення з'являється тільки в цьому вікні та не з'являється в іншій кімнаті. Потім другий користувач надсилає повідомлення в свою кімнату, і тест перевіряє, що це повідомлення з'являється тільки в цьому вікні та не з'являється в іншій кімнаті.

Лістинг 3.8– Дві кімнати

```
           def
test_when_chat_message_posted_then_not_seen_by_anyone_in_differe
nt_room(self):
           try:
                self._enter_room("user_1", "room_1")
                self._open_new_window()
                self._enter_room("user_2", "room_2")
                self._switch_to_window(0)
```

```
                self._send_msg("hello")
                WebDriverWait(self.driver, 2).until(
                     lambda _: "hello" in self._chat_log_value,
                     "Message was not received by window 1 from
window 1",
                )
                self._switch_to_window(1)
                self._send_msg("world")
                WebDriverWait(self.driver, 2).until(
                     lambda _: "world" in self._chat_log_value,
                     "Message was not received by window 2 from
window 2",
                )
                self.assertTrue(
                     "hello" not in self._chat_log_value,
                     "Message was improperly received by window 2
from window 1",
                )
           finally:
                self._close_all_new_windows()
```

```
% ./manage.py test video_player.tests
Found 2 test(s).
Creating test database for alias 'default'...
Destroying old test database for alias 'default'...
Type 'yes' if you would like to try deleting the test database '/home/rozhelluk/projects/wtg/src/db.sqlite3', or 'no' to cancel: yes
System check identified no issues (0 silenced).
room_1
room_2
.room_1
room_1
.
----------------------------------------------------------------------
Ran 2 tests in 6.649s

OK
Destroying test database for alias 'default'...
```

Рисунок 3.3 – Виконання тестів

Нефункціональне тестування, з іншого боку, зосереджене на аспектах якості програмного забезпечення, які не пов'язані з конкретними функціональними можливостями. Основна мета цього типу тестування – оцінка таких характеристик, як продуктивність, масштабованість, надійність, зручність використання, безпека, сумісність та інші. Нефункціональне

тестування допомагає визначити, наскільки добре програмне забезпечення відповідає очікуванням користувачів у реальних умовах експлуатації.

Було проведено тестування головної сторінки (рис 3.4). На ній можна зручно ввести своє ім'я та назву кімнати, до якої потрібно приєднатись. Поля вводу розміщені посередині сторінки та мають підказки.

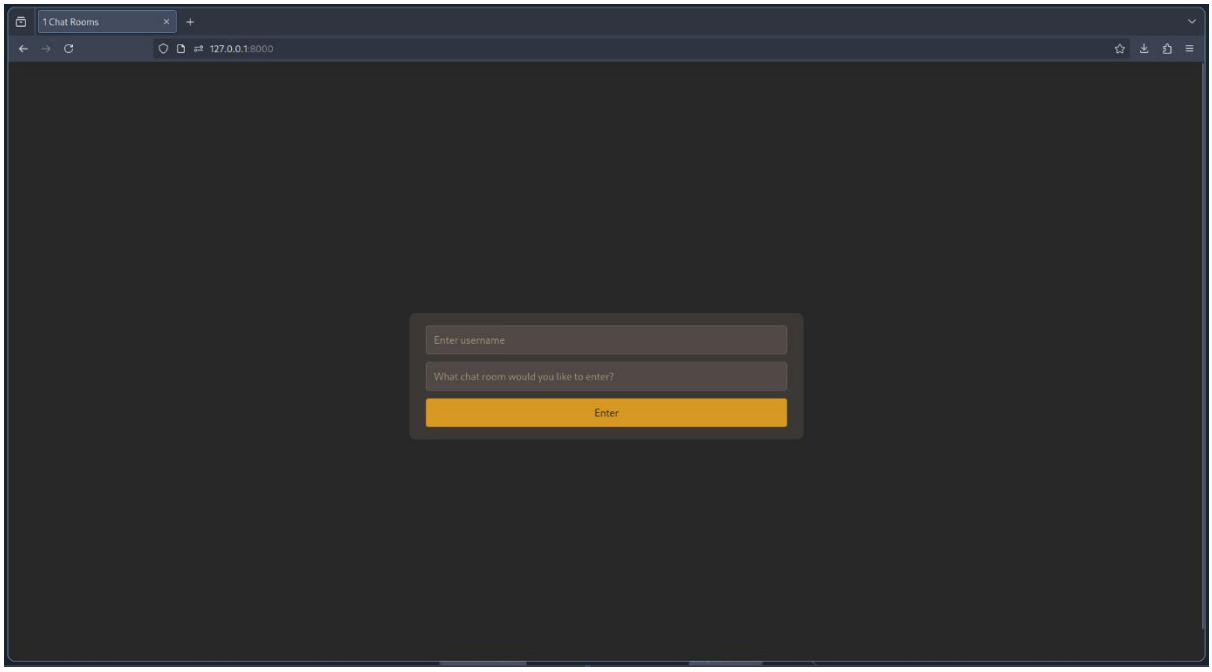

Рисунок 3.4 – Головна сторінка сайту

Також було протесто сторінку кімнати. На ній відображається відео та є зручний чат(рис 3.5).

Рисунок 3.5 – Сторінка кімнати

Дії з відео, а саме зупинка, відновлення, зміна часу відбувається синхронно у всіх користувачів(рис. 3.6), є можливість змінити гучність відео та відкрити його у повноекранному режимі.

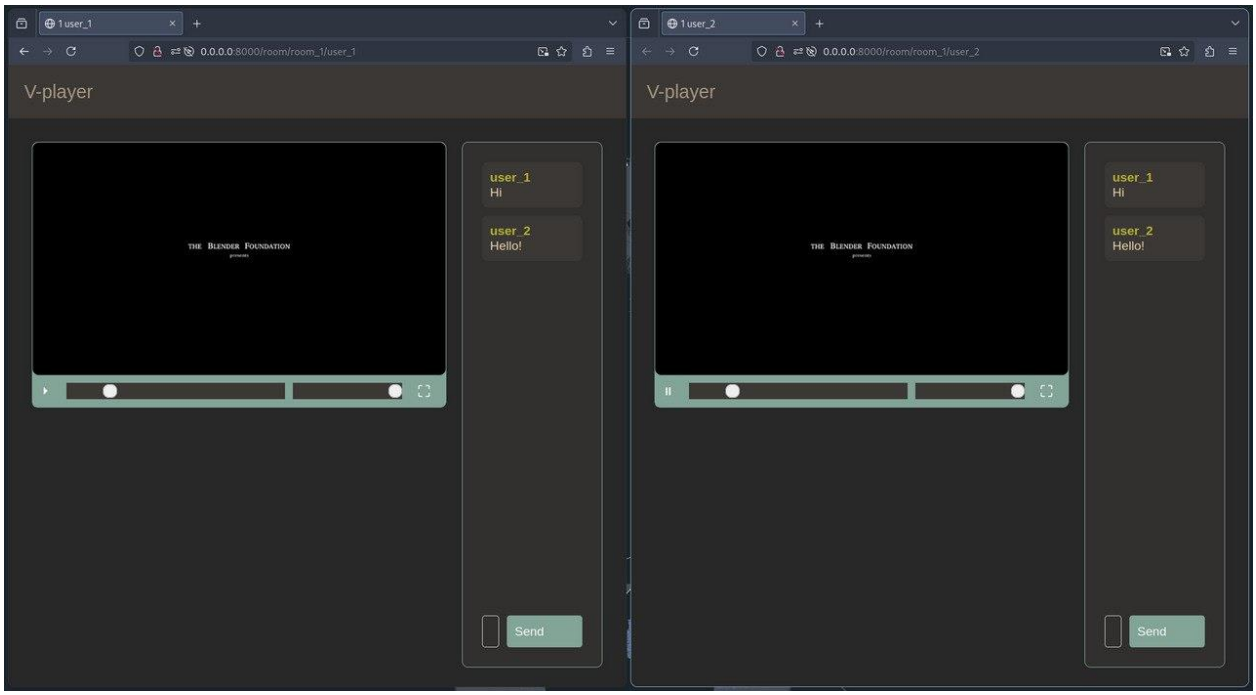

Рисунок 3.6 – Робота двох клієнтів

У чаті відображаються повідомлення, які мають текстовий зміст та ім'я користувача, при наведенні на кнопку "Send" курсор стає вказівником. Наступним кроком було проведення тестування надійності: при надсиланні великої кількості запитів, система встигала їх обробляти, зберігати потрібні дані в базі даних та давати відповідь. Жодних помилок виявлено не було.

Отже, у даному розділі було повністю створено та протестовано вебсистему для спільного перегляду та обговорення відеоконтенту. Також були проаналізовані програмне забезпечення та технології розробки та продемонстровано встановлення залежностей системи.

РОЗДІЛ 4 ТЕХНІКО-ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ

4.1 Аналіз ринку

У сучасному світі, вебзастосунки стали невід'ємною частиною світової мережі Інтернет. Вони включають різноманітні категорії, такі як інтернет-магазини, сайти з новинами, інтернет-банкінг, системи обліку товарів та бухгалтерії тощо. Щодня мільйони користувачів взаємодіють із цими застосунками, що підкреслює їх важливість і популярність.

Розроблений застосунок повинен надавати змогу користувачам створювати власні кімнати для перегляду відеоконтенту.

Основними рисами вебсистеми для спільного перегляду та обговорення є:

– Можливість перегляду з будь-якого пристрою що підтримує перегляд вебсайтів

– Мінімалістичний візуальний інтерфейс системи

– Можливість власноруч модифікувати систему

– Можливість розгортання продукту у власні мережі особистого та загального користування.

Водночас з розробкою продукту було проведено аналіз існуючих рішень, що доступні для загального використання на ринку. На них реалізований базовий функціонал, що є безплатним, та надається можливість отримати додатковий функціонал за плату.

Головною перевагою вебсистеми є можливість розгортання продукту у власні мережі, що дозволяє використовувати її не тільки для перегляду розважаючого відеоконтенту, але й для проведення дистанційного навчання, бізнес-конференцій тощо. Такий підхід забезпечує значну гнучкість у використанні, оскільки дозволяє адаптувати систему до конкретних потреб організації або користувача.

4.2 Розрахунок витрат на проєктування

Матеріальна частина охоплює витрати на розробку вебсистеми, грошову винагороду девелоперам тощо. Обсяг винагороди розробників залежить від ефективності роботи розробника, складності виконання поставленого завдання та наслідків зробленої діяльності та можливостей фірми.

Заробітна плата за виконання працівником роботи складається з двох частин: основна та неосновна, і залежить від якості реалізації поставленого завдання до моменту настання дедлайну. Вона може бути фіксованою як на весь проєкт, так і на його частини або таски. Додаткова або неосновна плата виплачується працівникам за виконання додаткових робіт, що першочергово не були пріоритетними.

Мінімальна заробітна плата з 01.04.2024 у місячному розмірі становить 8000 грн. Заробітна плата розробника не може бути нижчою від мінімальної зп.

Створення вебсистеми для спільного перегляду та обговорення відеоконтенту робили такі працівники: back-end developer, front-end developer, проєктувальник, QA engineer та дизайнер.

Back-end developer за повний відпрацьований місяць отримав 16000 гривень.

Рахуємо податок на доходи фізичних осіб: 16000 * 18% = 2880 гривень.

Рахуємо військовий збір: 16000 * 1,5% = 240 гривень.

Рахуємо єдиний внесок: 16000 * 22% = 3520 гривень.

Утримання: 2880 + 240 = 3120 гривень.

До виплати працівникові – 16000 – 3120 = 12880 гривень.

Front-end developer за повний відпрацьований місяць отримав 12000 гривень.

Рахуємо податок на доходи фізичних осіб: 12000 * 18% = 2160 гривень.

Рахуємо військовий збір: 12000 * 1,5% = 180 гривень.

Рахуємо єдиний внесок: 12000 * 22% = 2640 гривень.

Утримання: 2160 + 180 = 2340 гривень.

До виплати працівникові – 12000 – 2340 = 9660 гривень.

QA engineer за повний відпрацьований місяць отримав 11000 гривень.

Рахуємо податок на доходи фізичних осіб: 11000 * 18% = 1980 гривень

Рахуємо військовий збір: 11000 * 1,5% = 165 гривень.

Рахуємо єдиний внесок: 11000 * 22% = 2420 гривень.

Утримання: 1980 + 165 = 2145 гривень.

До виплати працівникові – 11000 – 2145 = 8855 гривень.

Проєктувальник за повний відпрацьований місяць отримав 14000 гривень.

Рахуємо податок на доходи фізичних осіб: 14000 * 18% = 2520 гривень.

Рахуємо військовий збір: 14000 * 1,5% = 210 гривень.

Рахуємо єдиний внесок: 14000 * 22% = 3080 гривень.

Утримання: 2520 + 210 = 2730 гривень.

До виплати працівникові – 14000 – 2730 = 11270 гривень.

Дизайнер за повний відпрацьований місяць отримав 10000 гривень.

Рахуємо податок на доходи фізичних осіб: 10000 * 18% = 1800 гривень.

Рахуємо військовий збір: 10000 * 1,5% = 150 гривень.

Рахуємо єдиний внесок: 10000 * 22% = 2200 гривень.

Утримання: 1800 + 150 = 1950 гривень.

До виплати працівникові – 10000 – 1950 = 8050 гривень

Результати обрахунків окладів зображені в таблиці 4.1.

Таблиця 4.1 – Відрахування заробітніх плат

№ п/п	Посада	Оклад, грн./міс	Відрахування, грн./міс	Кількість		Сума, грн
1	Back-end developer	16000	2280	1 чол.	5 міс.	65600
2	Front-end developer	12000	2160	1 чол.	5 міс.	49200

Продовження таблиці 4.1

3	QA engineer	11000	1980	1 чол.	1 міс.	9020
4	Проєктувальник	14000	2520	1 чол.	1 міс.	11480
5	Дизайнер	10000	1800	1 чол.	1 міс.	8200
		Усього:				143500

Сума відрахувань від зарплати на соціальні потреби складає: 3520+2640+2420+3080+2200 = 13860 грн.

Відрядження та контрагентські функції не проводилися, тому й розходів на це не було.

Інших прямих розходів також не було.

Загальна сума прямих розходів складає 13860 гривень.

Накладні витрати за один місяць складають 40% від суми прямих витрат – 13860 * 40% = 5544 гривені.

Планові накопичення складають 25% від суми прямих та накладних витрат – (13860 + 5544)*25% = 4851 гривню.

Кошторисна ціна додатка: 5544 + 13860 + 4851 = 24255 гривень.

Ціна за договором: 4851 + 24255 = 29106 гривень.

Податок на додаткову ціну: 24255*20% = 4851 гривеня.

Кошторис розходів у проєктуванні зведено у таблиці 4.2

Таблиця 4.2 – Кошторис витрат на проєктування

Найменування статей витрат	Сума, грн	Обґрунтування
1. Зарплата розробників	143500	
2. Відрахування на соціальні потреби.	13860	
3. Контрагентські роботи і послуги.	-	Не відбулися
4. Відрядження.	-	Не відбулися

Продовження таблиці 4.2

5. Інші прямі витрати	-	
6. Усього прямих витрат	13860	
7. Накладні витрати	5544	
8. Передбачуване зібрання	4851	
9. Загальна вартість проєкту	24255	
10. Податок на додаткову вартість	4851	
11. Загалом, ціна за договором	29106	

4.3 Обґрунтування необхідності

Розробка вебсервісу для спільного перегляду й обговорення відеоконтенту відповідає на кілька важливих потреб користувачів. Сучасні споживачі все більше орієнтуються на інтерактивність та соціальну складову використання інтернет-ресурсів. Спільний перегляд відео дозволяє користувачам не лише насолоджуватися контентом, але й обговорювати його в реальному часі з друзями чи колегами, що підвищує рівень взаємодії та задоволення від перегляду.

Економічний ефект від запровадження такого вебсервісу може бути значним. Завдяки інтерактивності та соціальній складовій сервіс привертає більше користувачів, що збільшує аудиторію. Зростання кількості користувачів, у свою чергу, підвищує потенціал для монетизації через рекламу, преміум-підписки та інші платні послуги. Економічні показники покращуються завдяки підвищенню середнього часу, проведеного користувачами на платформі, а також завдяки більшій лояльності користувачів, що сприяє зростанню доходів.

Основні напрямки отримання ефекту при впровадженні проєкту полягають у створенні комфортного та інтерактивного середовища для користувачів. Це включає в себе реалізацію функцій, які дозволяють

синхронізувати перегляд відео, організовувати чат у реальному часі та інші соціальні інтеракції. Крім того, сервіс може використовувати аналітичні інструменти для збору даних про поведінку користувачів, що дозволяє краще зрозуміти їхні уподобання та відповідно адаптувати контент і рекламні кампанії. У підсумку, ці заходи сприяють збільшенню доходів і зменшенню витрат на утримання та залучення користувачів, що створює значний економічний ефект для підприємства

Таким чином, розробка вебсервісу для спільного перегляду й обговорення відеоконтенту принесе користь широкому колу користувачів – від індивідуальних глядачів до навчальних закладів, корпоративних клієнтів, контент-провайдерів та рекламодавців.

ВИСНОВКИ

Під час виконання кваліфікаційній роботі було розроблено вебсистему для спільного перегляду та обговорення відеоконтенту, яка надає користувачам можливість взаємодіяти з відеоматеріалами у режимі реального часу. Метою було створити платформу, що дозволяє користувачам синхронно переглядати відео та обговорювати його через інтегрований чат, забезпечуючи тим самим новий рівень інтерактивності та спільного досвіду.

Розробка цієї системи включає детальний аналіз існуючих рішень, визначення їх переваг та недоліків, що дозволило сформулювати основні вимоги до нової платформи. Було розглянуто та впроваджено ряд технічних рішень для забезпечення синхронізації відео, стабільної роботи чату та зручного користувацького інтерфейсу. Технічні аспекти реалізації включали використання сучасних вебтехнологій, таких як WebSockets для забезпечення миттєвого обміну повідомленнями та забезпечення синхронного відтворення відеоматеріалу. Також було використано Redis– програмне забезпечення для зберігання даних про відео в пам'яті .

Результати тестування показали, що розроблена система відповідає поставленим вимогам та успішно вирішує основні завдання. Система забезпечує стабільну синхронізацію відео для всіх учасників сеансу, надає можливість взаємодії через інтегрований чат та забезпечує високий рівень зручності для користувачів. Важливим результатом є також те, що платформа може бути легко адаптована для різних аудиторій та використання в різних контекстах: від розважальних до освітніх.

Подальший розвиток проєкту може включати розширення функціональності системи, зокрема, додавання підтримки різних типів відеоконтенту, інтеграцію з соціальними мережами та іншими онлайн-сервісами, а також покращення користувацького інтерфейсу. Крім того, можливим напрямком розвитку є оптимізація системи для роботи з великими

групами користувачів, що дозволить використовувати її у масових онлайн-заходах та трансляціях.

Таким чином, розробка вебсистеми для спільного перегляду та обговорення відеоконтенту є важливим кроком у напрямку створення нових інструментів для інтерактивного та колективного споживання медіа. Вона відкриває нові можливості для взаємодії користувачів та спільного досвіду, що робить її цінним внеском у розвиток сучасних цифрових технологій та інтернет-сервісів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Django. *Вікіпедія*: вебсайт. URL: https://uk.wikipedia.org/wiki/Django (дата звернення: 13.05.2024).

2. Arch Linux. *Вікіпедія*: вебсайт. URL: https://uk.wikipedia.org/wiki/Arch_Linux (дата звернення: 12.05.2024).

3. Встановлення Neovim. *Документація Rockylinux*: вебсайт. URL: https://docs.rockylinux.org/uk/books/nvchad/install_nvim (дата звернення: 08.05.2024).

4. WebSocket. *Вікіпедія*: вебсайт. URL: https://uk.wikipedia.org/wiki/WebSocket (дата звернення: 15.05.2024).

5. Python. *Вікіпедія*: вебсайт. URL: https://uk.wikipedia.org/wiki/Python (дата звернення: 13.05.2024).

6. Redis. *Вікіпедія*: вебсайт. URL: https://uk.wikipedia.org/wiki/Redis (дата звернення: 09.05.2024).

7. Docker. *Вікіпедія*: вебсайт. URL: https://uk.wikipedia.org/wiki/Docker (дата звернення: 14.05.2024).

8. Git. *Вікіпедія*: вебсайт. URL: https://uk.wikipedia.org/wiki/Git (дата звернення: 01.03.2024).

ДОДАТКИ

Додаток А

Лістинг клієнтської частини

```
    const roomName =
JSON.parse(document.getElementById("room-
name").textContent);
    const socket = new WebSocket(
      "ws://" + window.location.host + "/ws/room/" +
roomName + "/",
    );
    socket.onopen = function (e) {
      console.log("WebSocket open!", e);
    };
    socket.onclose = function (e) {
      console.error("Chat socket closed unexpectedly",
e);
    };
    const video = document.getElementById("video");
    const playPauseButton =
document.getElementById("playPauseButton");
    const timeSlider =
document.getElementById("timeSlider");
    const volumeSlider =
document.getElementById("volumeSlider");
    const fullscreenButton =
document.getElementById("fullscreenButton");
    playPauseButton.addEventListener("click", function
() {
      if (video.paused) {
        video.play();
        playPauseButton.value = "\u23F8";
      } else {
        video.pause();
        playPauseButton.value = "\u23F5";
      }
    });
    video.addEventListener("timeupdate", function () {
      const value = (100 / video.duration) *
video.currentTime;
      timeSlider.value = value;
    });
```

```
    timeSlider.addEventListener("input", function () {
      const time = (timeSlider.value / 100) *
video.duration;
      video.pause();
      video.currentTime = time;
    });
    volumeSlider.addEventListener("input", function ()
{
      video.volume = volumeSlider.value / 100;
    });
    fullscreenButton.addEventListener("click", function
() {
      if (!document.fullscreenElement) {
        if (video.requestFullscreen) {
          video.requestFullscreen();
        } else if (video.mozRequestFullScreen) {
          video.mozRequestFullScreen();
        } else if (video.webkitRequestFullscreen) {
          video.webkitRequestFullscreen();
        } else if (video.msRequestFullscreen) {
          video.msRequestFullscreen();
        }
      } else {
        if (document.exitFullscreen) {
          document.exitFullscreen();
        } else if (document.mozCancelFullScreen) {
          document.mozCancelFullScreen();
        } else if (document.webkitExitFullscreen) {
          document.webkitExitFullscreen();
        } else if (document.msExitFullscreen) {
          document.msExitFullscreen();
        }}
    });
    socket.onmessage = function (event) {
      const data = JSON.parse(event.data);
      if (data.type === "video.pause" || data.type ===
"video.play") {
        handleVideoPaused(data);
      } else if (data.type === "get.video.status") {
        getVideoStatus();
      } else if (data.type === "set.video.status") {
        setVideoStatus(data);
      } else if (data.type === "video.seeked") {
```

```
        if (Math.trunc(video.currentTime) !==
Math.trunc(data.currentTime)) {
          video.currentTime = data.currentTime;
        }
      } else if (data.type === "chat.message") {
        handleMessage(data);
      } else if (data.type === "error") {
        console.error("Error:", data.message);
      } else {
        console.warn("Unknown message type:",
data.type, "data:", data);
      }
    };
    function setVideoStatus(data) {
      const video = document.getElementById("video");
      video.src = data.src;
      video.currentTime = data.currentTime;
      video.paused = data.paused;
    }
    function getVideoStatus() {
      socket.send(
        JSON.stringify({
          type: "get.video.status",
          paused: video.paused,
          currentTime: video.currentTime,
          src: video.src,
        }),
      );
    }
    function handleVideoPaused(data) {
      if (data.paused === 1 || video.paused === 0) {
        video.pause();
      } else if (data.paused === 0 || video.paused ===
1) {
        video.play();
      }
    }
    function handleMessage(data) {
      const regex =
        /https?:\/\/(www\.)?[-a-zA-Z0-
9@:%._\+~#=]{1,256}\.[a-zA-Z0-9()]{1,6}\b([-a-zA-Z0-
9()@:%_\+.~#?&//=]*)/g;
      const matches = data.message.match(regex);
      if (matches) {
        socket.send(
```

```
          JSON.stringify({
            type: "set.video.src",
            src: data.message,
          }),
        );
        video.src = data.message;
        data.message = "New video src = " +
data.message;}
      const message_content =
document.createElement("div");
      message_content.className = "message-content";
      message_content.innerText = data.message;

      const username = document.createElement("div");
      username.className = "username";
      username.innerText = data.username;
      const message = document.createElement("div");
      message.className = "message";
      message.appendChild(username);
      message.appendChild(message_content);

document.querySelector("#messages").appendChild(message
);
      const messages =
document.getElementById("messages");
      messages.scrollTop = messages.scrollHeight;
    }
    document.querySelector("#input-message").focus();
    document.querySelector("#input-message").onkeyup =
function (e) {
      if (e.key === "Enter") {
        document.querySelector("#send-
message").click();
      }
    };
    document.querySelector("#send-message").onclick =
function (e) {
      const messageInputDom =
document.querySelector("#input-message");
      const message = messageInputDom.value;
      const username =
JSON.parse(document.getElementById("username").textCont
ent);
      if (message.trim() === "") return;
      socket.send(
```

```
      JSON.stringify({
        type: "chat.message",
        message: message,
        username: username,
      }),
    );
    messageInputDom.value = "";
  };
  video.addEventListener("play", function () {
    socket.send(
      JSON.stringify({
        type: "video.play",
        paused: 0,
      }),
    );
  });
  video.addEventListener("pause", function () {
    socket.send(
      JSON.stringify({
        type: "video.pause",
        paused: 1,
      }),
    );
  });
  video.addEventListener("timeupdate", function () {
    socket.send(
      JSON.stringify({
        type: "video.timeupdate",
        currentTime: video.currentTime,
      }),
    );
  });
  video.addEventListener("seeked", function () {
    isSeeking = true; // Set the flag when seeking
    socket.send(
      JSON.stringify({
        type: "video.seeked",
        currentTime: video.currentTime,
      }),
    );
  });
```

Додаток Б

Лістинг серверної частини

```
    import json
    import redis.asyncio as redis
    from channels.generic.websocket import
AsyncWebsocketConsumer
    class Consumer(AsyncWebsocketConsumer):
        async def get_redis_video_status(self):
            """Get info about video from Redis"""
            redis_conn = redis.Redis(host="localhost",
port=6379, db=0)
            paused = await
redis_conn.get(f"{self.room_group_name}_video_paused")
            currentTime = await
redis_conn.get(f"{self.room_group_name}_video_currentTi
me")
            src = await
redis_conn.get(f"{self.room_group_name}_video_src")
            await redis_conn.close()
            return (paused, currentTime, src)
        async def set_redis_video_status(self, paused,
currentTime, src):
            """Set info about video to Redis"""
            redis_conn = redis.Redis(host="localhost",
port=6379, db=0)
            await
redis_conn.set(f"{self.room_group_name}_video_paused",
int(paused))
            await redis_conn.set(

f"{self.room_group_name}_video_currentTime",
float(currentTime)
            )
            await
redis_conn.set(f"{self.room_group_name}_video_src",
str(src))
            await redis_conn.close()

        async def set_redis_video_src(self, src):
            """Set video.src to Redis"""
            redis_conn = redis.Redis(host="localhost",
port=6379, db=0)
```

```
            await
redis_conn.set(f"{self.room_group_name}_video_paused",
0)
            await
redis_conn.set(f"{self.room_group_name}_video_currentTi
me", 0)
            await
redis_conn.set(f"{self.room_group_name}_video_src",
src)
            await redis_conn.close()
        async def set_redis_video_timeupdate(self,
currentTime):
            """Set video.src to Redis"""
            redis_conn = redis.Redis(host="localhost",
port=6379, db=0)
            await
redis_conn.set(f"{self.room_group_name}_video_currentTi
me", currentTime)
            await redis_conn.close()

        async def set_redis_video_paused(self, paused):
            """Set video.src to Redis"""
            redis_conn = redis.Redis(host="localhost",
port=6379, db=0)
            await
redis_conn.set(f"{self.room_group_name}_video_paused",
paused)
            await redis_conn.close()
        @staticmethod
        def validate_video(v_paused, v_currentTime,
v_src):
            if v_paused is None:
                paused = 0
            else:
                paused = v_paused.decode("utf-8")
            if v_currentTime is None:
                currentTime = 0
            else:
                currentTime =
v_currentTime.decode("utf-8")
            if v_src is None:
                src =
"/static/video_player/media/trailer.mp4"
            else:
                src = v_src.decode("utf-8")
```

```
            return (paused, currentTime, src)
        async def connect(self):
            self.room_name =
self.scope["url_route"]["kwargs"]["room_name"]
            self.room_group_name =
f"chat_{self.room_name}"
            await self.channel_layer.group_add(
                self.room_group_name,
                self.channel_name,
            )
            paused, currentTime, src = await
self.get_redis_video_status()
            await self.accept()
            if None in (paused, currentTime, src):
                n_paused, n_currentTime, n_src =
self.validate_video(
                    paused, currentTime, src
                )
                await self.send(
                    text_data=json.dumps(
                        {
                            "type": "set.video.status",
                            "paused": n_paused,
                            "currentTime":
n_currentTime,
                            "src": n_src,
                        }
                    )
                )
            else:
                await self.send(
                    text_data=json.dumps(
                        {
                            "type": "set.video.status",
                            "paused":
paused.decode("utf-8"),
                            "currentTime":
float(currentTime.decode("utf-8")),
                            "src": src.decode("utf-8"),
                        }
                    )
                )
        async def receive(self, text_data):
            try:
                text_data_json = json.loads(text_data)
```

```
            except json.JSONDecodeError:
                await self.send_error_message("Invalid
JSON format")
                return
            if "type" not in text_data_json:
                await self.send_error_message("Invalid
message type")
                return
            msg_type = text_data_json["type"]
            if msg_type == "video.pause":
                await
self.set_redis_video_paused(text_data_json["paused"])
                await self.channel_layer.group_send(
                    self.room_group_name,
                    {
                        "type": msg_type,
                        "paused":
text_data_json["paused"],
                    },
                )
            elif msg_type == "get.video.status":
                await self.set_redis_video_status(
                    text_data_json["paused"],
                    text_data_json["currentTime"],
                    text_data_json["src"],
                )
            elif msg_type == "set.video.src":
                await
self.set_redis_video_src(text_data_json["src"])
            elif msg_type == "video.timeupdate":
                await
self.set_redis_video_timeupdate(text_data_json["current
Time"])
            elif msg_type == "video.seeked":
                await self.channel_layer.group_send(
                    self.room_group_name,
                    {
                        "type": msg_type,
                        "currentTime":
text_data_json["currentTime"],
                    },
                )
            elif msg_type == "chat.message":
                await self.channel_layer.group_send(
                    self.room_group_name,
```

```
                    {
                        "type": msg_type,
                        "message":
text_data_json["message"],
                        "username":
text_data_json["username"],
                    },
                )
            elif msg_type == "video.play":
                # 2
                await
self.set_redis_video_paused(text_data_json["paused"])

                await self.channel_layer.group_send(
                    self.room_group_name,
                    {
                        "type": msg_type,
                        "paused":
text_data_json["paused"],
                    },
                )
            else:
                await self.send_error_message("invalid
message format")
        async def disconnect(self, close_code):
            await self.channel_layer.group_discard(
                self.room_group_name,
                self.channel_name,
            )
        async def send_error_message(self,
error_message):
            await self.send(
                text_data=json.dumps(
                    {
                        "type": "error",
                        "message": error_message,
                    }
                )
            )
        async def chat_message(self, data):
            message = data["message"]
            username = data["username"]
            await self.send(
                text_data=json.dumps(
                    {
```

```
                    "type": "chat.message",
                    "message": message,
                    "username": username,
                }
            )
        )
    async def video_pause(self, data):
        # 3
        paused = data["paused"]
        await self.send(
            text_data=json.dumps(
                {
                    "type": "video.pause",
                    "paused": paused,
                }
            )
        )
    async def video_play(self, data):
        paused = data["paused"]
        await self.send(
            text_data=json.dumps(
                {
                    "type": "video.play",
                    "paused": paused,
                }))
    async def set_video_src(self, data):
        src = data["src"]
        await self.send(
            text_data=json.dumps(
                {
                    "type": "set.video.src",
                    "src": src,
                }))
    async def video_seeked(self, data):
        currentTime = data["currentTime"]
        await self.send(
            text_data=json.dumps(
                {
                    "type": "video.seeked",
                    "currentTime": currentTime,},))
```